Outside X68000

桒野雅彦 *Masahiko Kuwano* ●著

SOFT BANK

Outside X68000

桒野雅彦 Masahiko Kuwano ●著

はじめに

『Inside X68000』に続き，『Outside X68000』を刊行できることになりました。『Inside』がソフトウェア寄りの立場であったのに対して，本書はよりハードウェアに近い視点からX68000というパーソナルコンピュータを見つめてみたものです。

パソコンがより一般的な商品となるにつれて，ハードウェアは次第にブラックボックスとなってしまいました。パソコン雑誌にハードウェアに触れた記事があったり，ハードウェアと名がつく本が刊行されても，それらはあくまでもレジスタの配置やメモリマップといった，ソフトから見たときのものばかりで，ハードウェアがどのように組まれているのかといったことにはほとんど触れられていません。ソフトの立場から見る限り，ハードウェアというのはLSIのレジスタやV-RAM，メインメモリといったものであり，その先がどのようになっているかについてはほとんど気にする必要がないということなのでしょうか。

しかし，コンピュータの現実の動作というのはLSIの向こう側の動きに他なりません。ジョイスティックポートの読み出し方がわかったとしても，それだけではジョイスティックポートを使うような周辺機器の作り方がわかるわけではありません。

自作のオプション機器やボードを作ろうとすると，接続しようとしているコネクタや拡張スロットに出ている信号の意味だけでなく，それらの信号がどのような回路で作られ，どのようなICとつながっているのかといったことまで気を使う必要もあるでしょう。

LSIの信号ピンを外部回路によって細工しているような場合，ポートの説明はわかりにくくなりがちですし，LSIのマニュアルも読み変えなくてはならないこともあります。また，細工が行われていると，レジスタの説明などにはまったく出てこないような副作用に悩まされることもあります。

パーソナルコンピュータが真にパーソナルなものという立場に立ち，ユーザに解放されたものであるためにはどうしてもハードウェアに関する情報が必要といえるでしょう。

本書は，X68000の初代機とXVI，シャープ純正のオプションボード類の全回路図やインタフェース部の回路，ブロック図，ピン設定などをまとめるとともに，『Oh!X』誌に掲載した周辺機器の製作記事を集めました。

本書が，X68000のハードウェアを活用，解析するための資料として活用され，その能力を100%以上まで引き出す役に立てることを願ってやみません。

1993年2月

桒野雅彦

CONTENTS

ユニバーサルI/Oボード(CZ-6BU1) …… 175

●自作周辺機器編 …… 185

自作周辺機器製作例 …… 187

乱数発生機 …… 189

COVER DESIGN……Masaki KATSUMATA

本体編

X68000の内部回路

X68000の内部回路はかなり複雑なつくりになっています。この章では，X68000に用意されたさまざまな入出力コネクタまわりの回路と，主要ゲートアレイの変遷の2点についてまとめました。

1 はじめに

ハードウェア，特にデジタル回路を学ぶ上で，パーソナルコンピュータの内部回路というのは参考になるところが多いものです。特にX 68000はパーソナルコンピュータのなかでも多くの機能を内蔵しており，インタフェースの作り方や設計思想など，ちょっと調べるだけでもデジタル回路の実践資料として十分役にたつと思います。

この章では，X 68000の回路から各インタフェース部分のダイジェスト，機種ごとのハードウェアの変遷をまとめました。

●2 入出力部の回路

周辺機器を自作してX 68000と接続する場合，接続した信号の先がどのような回路になっているかということはぜひとも知っておきたい情報です。本体側の回路によって，その信号を周辺機器側でどのような回路で受けなくてはならないかといったことや，どのようなレベルの信号を与えればよいかといったことを考えることができます。

本書にはX 68000の回路図を載せていますから，必要とあらばLSIのピンの根元まで追いかけることができますが，高機能な本体だけあって回路図は相当に入り組んでおり，コネクタに付いているICまでたどりつくだけでも結構な労力を要します。そこで，ここでは拡張スロットやプリンタコネクタなど，ユーザが使用することができるI/Oコネクタの先がどのような回路となっているかを，初代機の回路図をもとにしてまとめてみました。抵抗やコンデンサの値など，細かい点は機種によって異なっている場合もありますが，互換性については十分考慮されていますので，初代機に基づいて作っておいた周辺機器がそれ以降の機種で動作しなくなるといったことはないはずです。どうしても気になる場合には，巻末に用意したXVIの回路図にもあたるようにしてください。

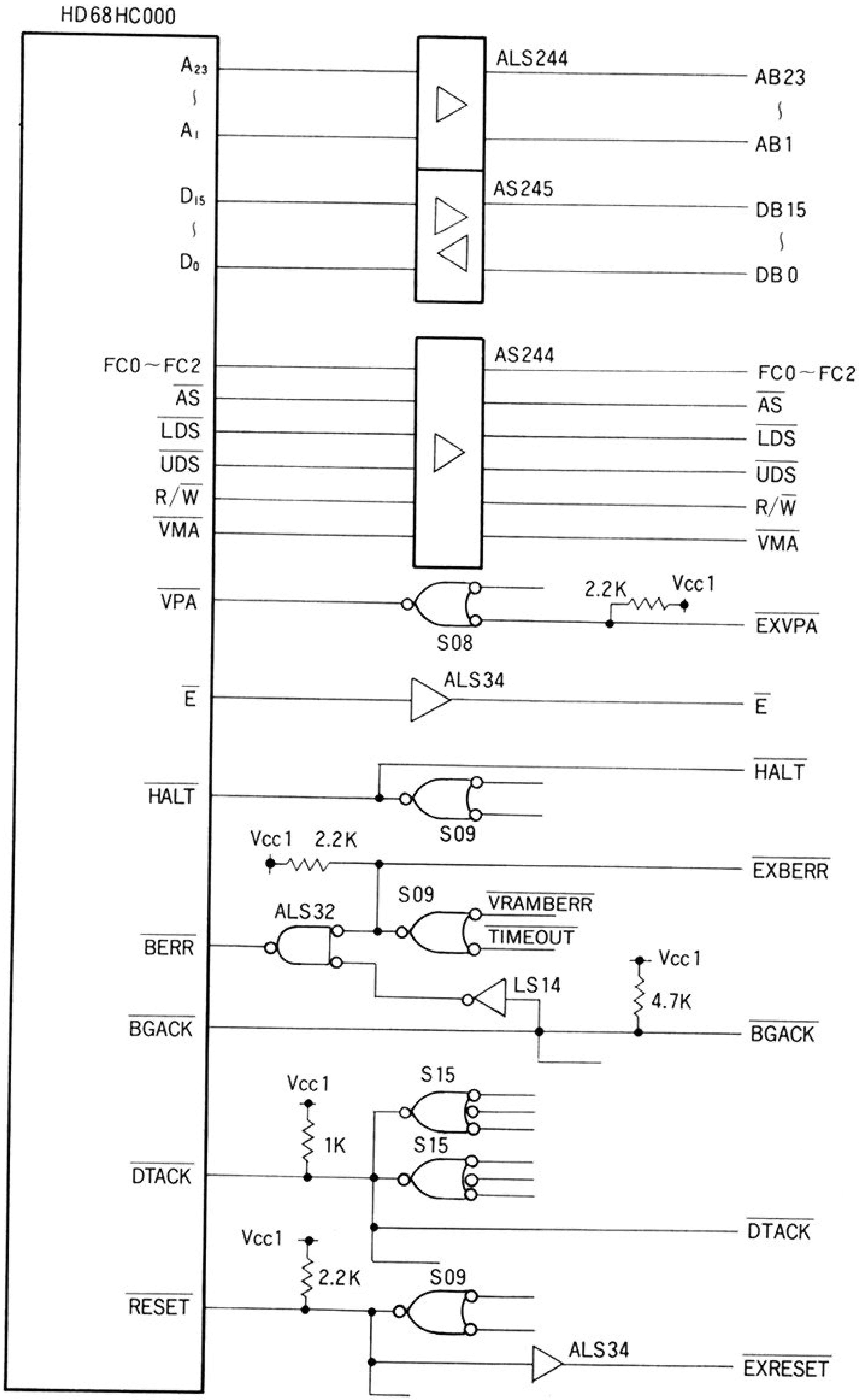
❷·1 拡張スロット

拡張スロットにはごく基本的なメモリやI/Oのリード/ライト動作のための信号のほか，クロック，DMAや割り込み要求のための信号，水平同期や垂直同期信号など何種類かの信号が混在して配置されており，さまざまな種類のボードを作ることができるようになっています。

❷·❶1 CPU関連の信号

拡張スロットの各信号のうち，CPUに直接接続されるものの系統を図1に示します。CPUと直接関係する信号である，アドレス，データ，ステータス信号などは，CPUに入っている信号をそのまま出力していたり，ALS（アドバンスト・ローパワー・ショットキー）やAS（アドバンスト・ショットキー），S（ショットキー）シリーズといった，高速TTLファミリーのICが一段入るだけにして，ゲート遅れによるタイミングのずれやタイミングマージンの減少を

●図……1　拡張スロットの信号（1）　CPU周辺



最小限に抑えるようにしています。

この部分の回路から，HALT，BGACK，EXBERR，DTACK 信号を拡張ボード側でドライブする場合には，オープンコレクタゲートを使うか，またはトライステートバッファを使って自分自身へのアクセスのとき以外はハイ・インピーダンスとするようにしなくてはならないこと，入力として用いる場合にはシュミットタイプを使った方がよさそうだということがわかります。

❷・❶2 バス解放要求，割り込み，同期信号

バス解放要求，割り込み，同期信号関係の回路を図 2 に示します。拡張スロットの各信号のうちバス解放要求信号と割り込み要求信号，NMI（マスク不可割り込み），IDDIR（バスバッファの方向を示す信号）はシャープ開発のゲートアレイの 1 つ，BUDDHA が受けるようになっています。バス解放要求信号（BRn，BGn）や割り込み要求関係の信号（IRQ 2, IRQ 4, IACK 2, IACK 4）はスロットごとに独立して持っており，BUDDHA で合成されるようになっています。BUDDHA が受ける信号のうち，IDDIR 以外の出力信号は拡張スロットに直結され，入力信号は拡張ボードがないときや，これらの信号を使わないときにハイ・インピーダンス状態になるのを避けるために 2.2 KΩ の抵抗でプルアップされています。これらの信号をオープンコレクタでドライブすることもできそうですが，立ち上がり波形がなまりやすいこと，せっかく拡張スロットごとに分離されて配線されていることを考えると，トーテムポール出力の IC を使用してドライブする方がよいでしょう。

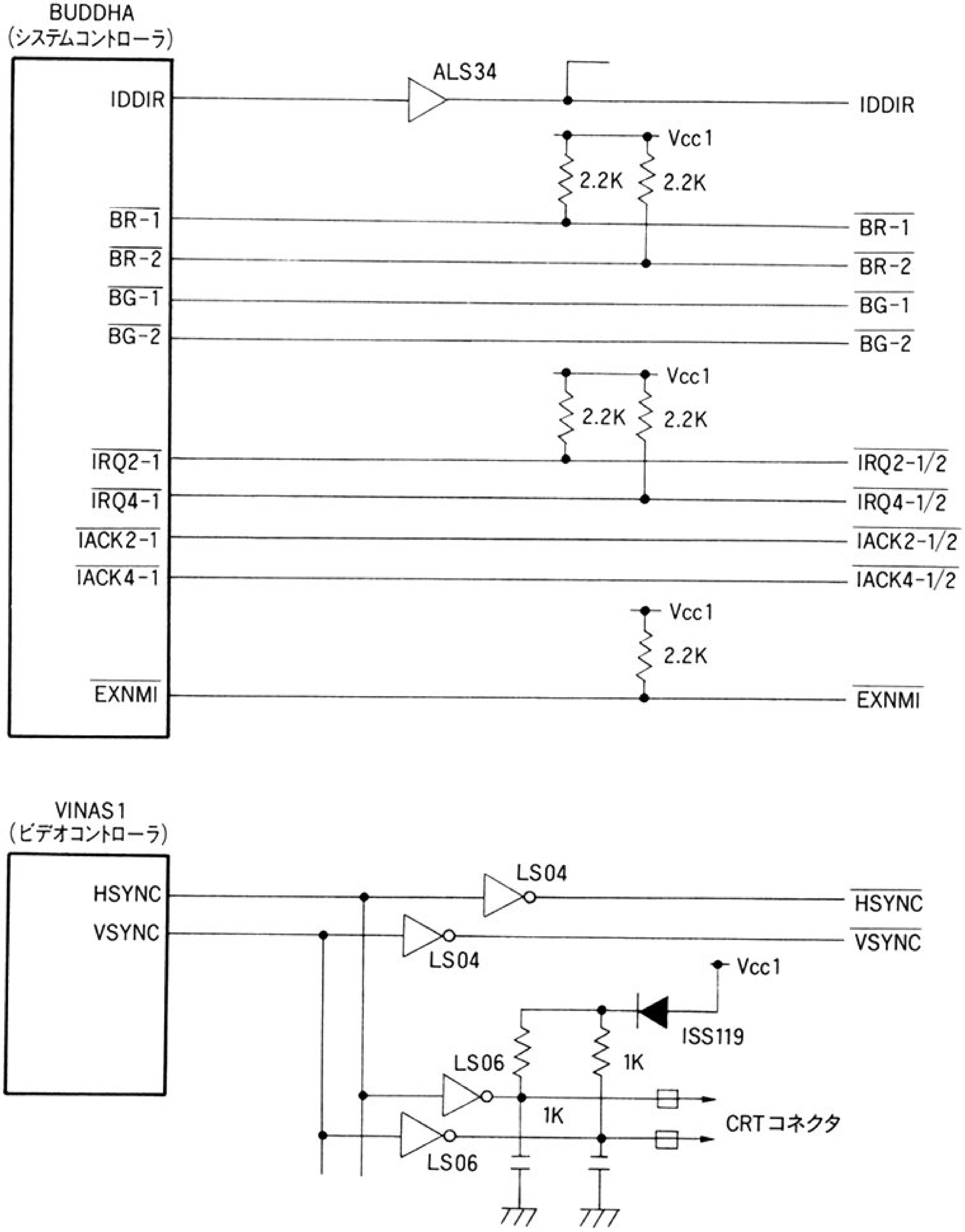
垂直，水平同期信号はビデオコントローラである，VINAS 1 が出力しています。CRT コネクタの方がオープンコレクタバッファを使っているのに対して，拡張スロットの方は LS 04 で出力しているなど，若干の違いはありますが，基本的には CRT コネクタに出力されている同期信号と同じものであるといえます。

❷・❶3 その他の信号

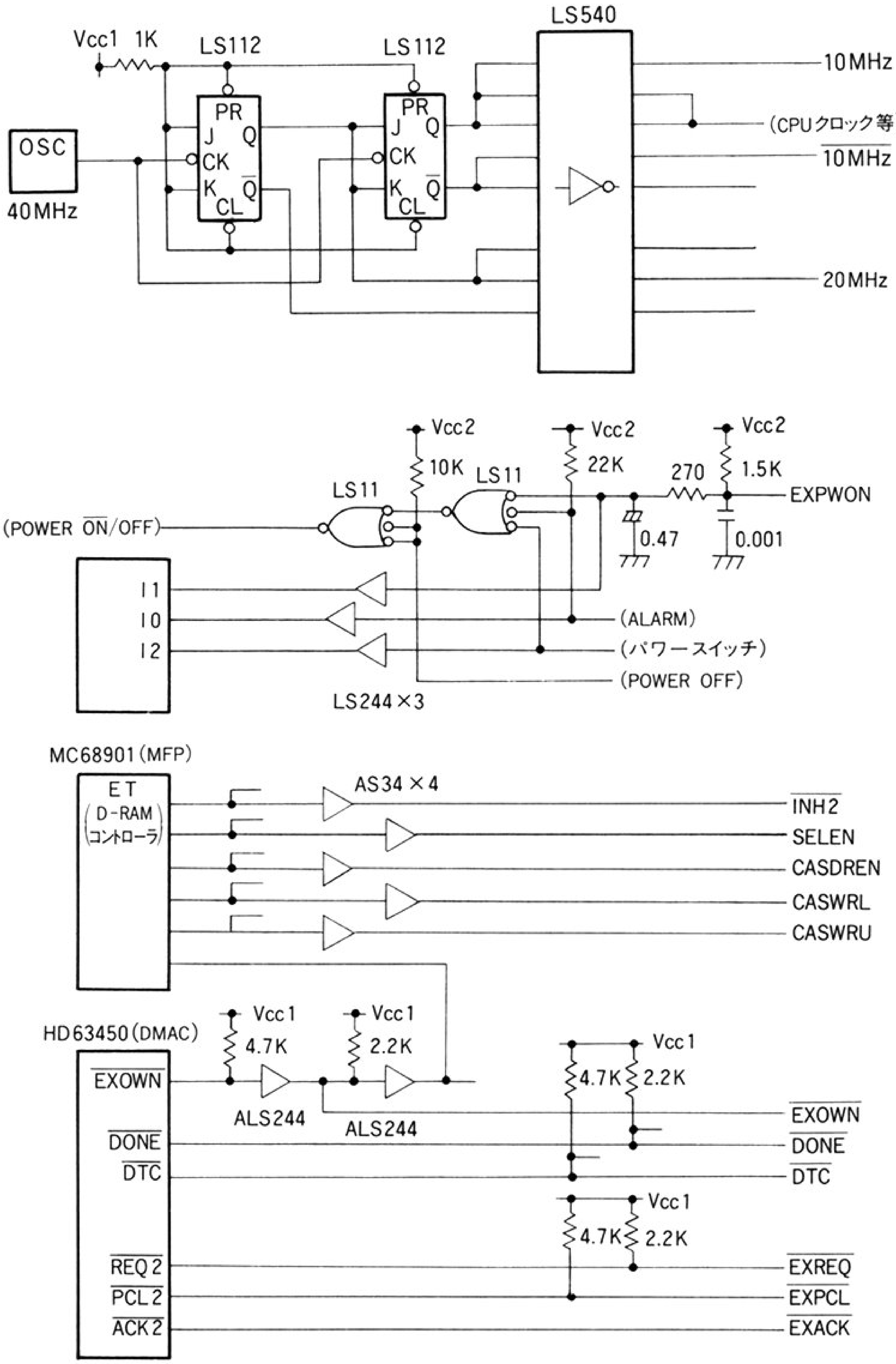
クロック関係，外部パワーオン信号，DMA 要求信号などの配置は図 3 のようになっています。クロックは 40 MHz の原発振を分周して得られた 20 MHz と 10 MHz を LS 540 でバッファして CPU のクロックや拡張スロットの各クロックなどとして使用するようにしています。

分周には JK フリップフロップが使用されており，20MHz と 10MHz のクロックの位相

●図……2　拡張スロットの信号（2）　バス交換，同期信号関係



●図……3　拡張スロットの信号（3）　クロック，DMA 要求等



関係がくずれないよう考慮されています。

EXPONはX 68000の電源をONにするための信号で，'L'レベルにすることで，本体正面の電源スイッチがONになっていなくても本体の電源を入れることができるようになっています。ノイズによる誤動作を防止するためか，CRによるフィルターが組まれています。X 68000のパワーON要因は，正面の電源スイッチとRTCのアラーム機能，そして外部パワーON信号の3つがあるため，どの要因で電源が投入されたのかをMFPのGPIP端子経由で読めるようにしています。

INH 2, SELEN, CASDREN, CASWRL, CASWRUの5つの信号はD-RAMコントローラである，ETが出力している信号です。これらの信号を使えば，D-RAMなどいとも簡単に接続できそうなのですが，実際の拡張メモリボードで使われているのは，リフレッシュサイクルであることを示すINH 2だけで，その他の信号はボード内でディレイラインなどを使って作り直しています。

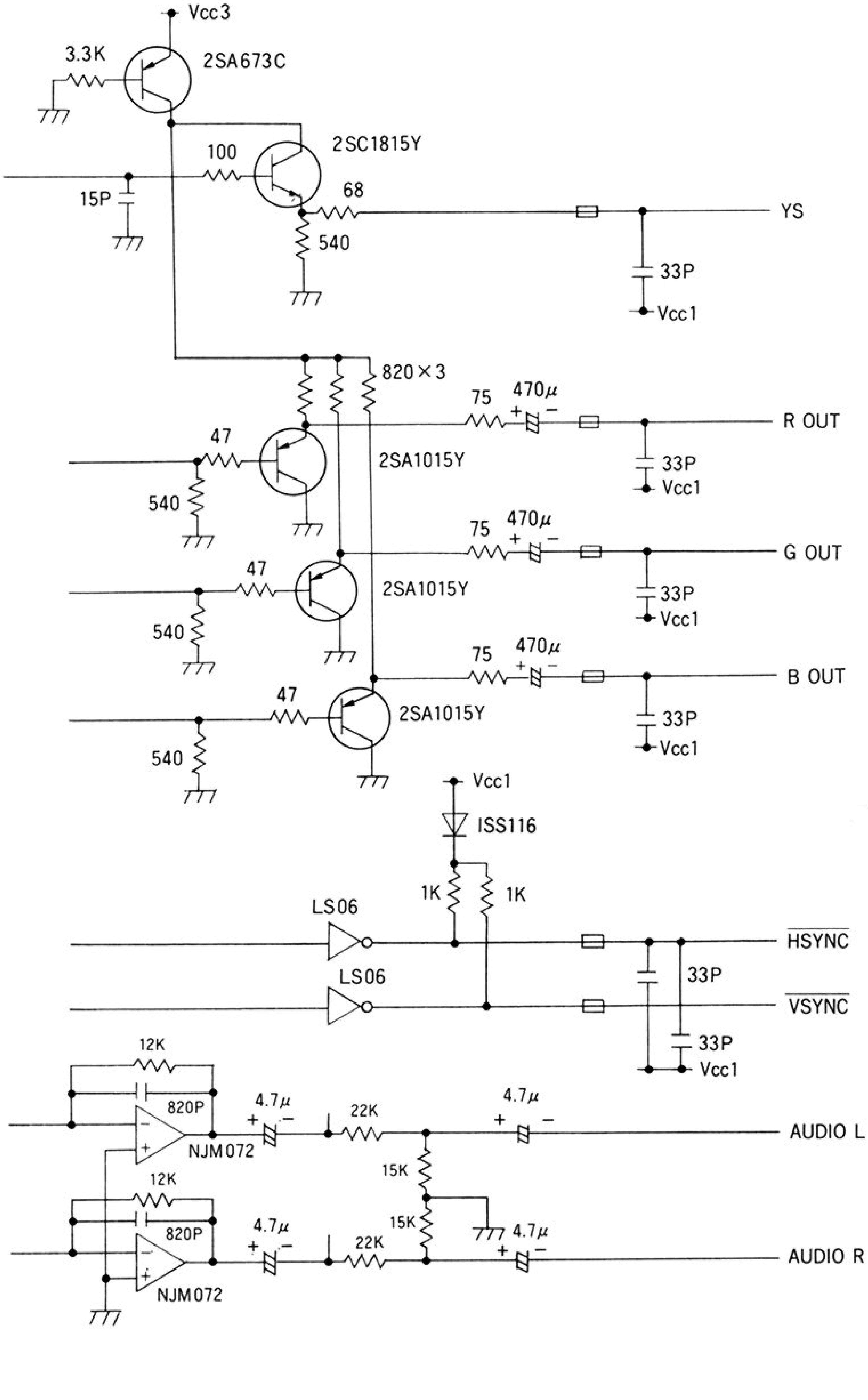
EXOWN, DONE, DTC, EXREQ, EXPCL, EXACKはDMAコントローラと接続される信号群です。DMAコントローラのチャンネル#2はオプションボード用に解放されており，これらの信号の使用によって，拡張スロットのうち1スロットはDMAを使用することができるようになっています。SUPER以前の機種にSCSIインタフェースをサポートするCZ-6 BS 1はこのDMAチャンネルを使用せず，標準装備であるSASIインタフェースの方で使用しているチャンネル#1を，動作モードの変更によって共用するという巧妙な方法をとったため，SCSIボードを入れても，貴重なDMAチャンネルは解放されたままになっています。

❷・2 アナログRGBコネクタ

アナログRGBコネクタ付近の回路は図4のようになっています。アナログRGBコネクタには，通常のRGB信号や同期信号のほか，左右の音声出力が配線されており，ディスプレイ側でステレオ対応ができるようにしています。RGBコネクタへの音声出力はNJM 072によるバッファアンプを通った後で，ヘッドフォンや内部スピーカなどと分岐します。音量調整はこの出力の後段，ヘッドフォン/スピーカ用のアンプの手前にありますので，アナログRGBコネクタへの出力レベルは本体前面の音量調整つまみの影響を受けません。

映像関係のRGB, YS, 同期信号はいずれもフェライトビーズと33 pFのコンデンサによるフィルターを通して出力されます。RGBの各映像出力は2 SA1015Y，YSは2 SC1815Yによるエミッタフォロワ，同期信号はオープンコレクタのLS06と1 KΩのプルアップ抵抗という構成になっています。

●図……4　アナログ RGB コネクタ



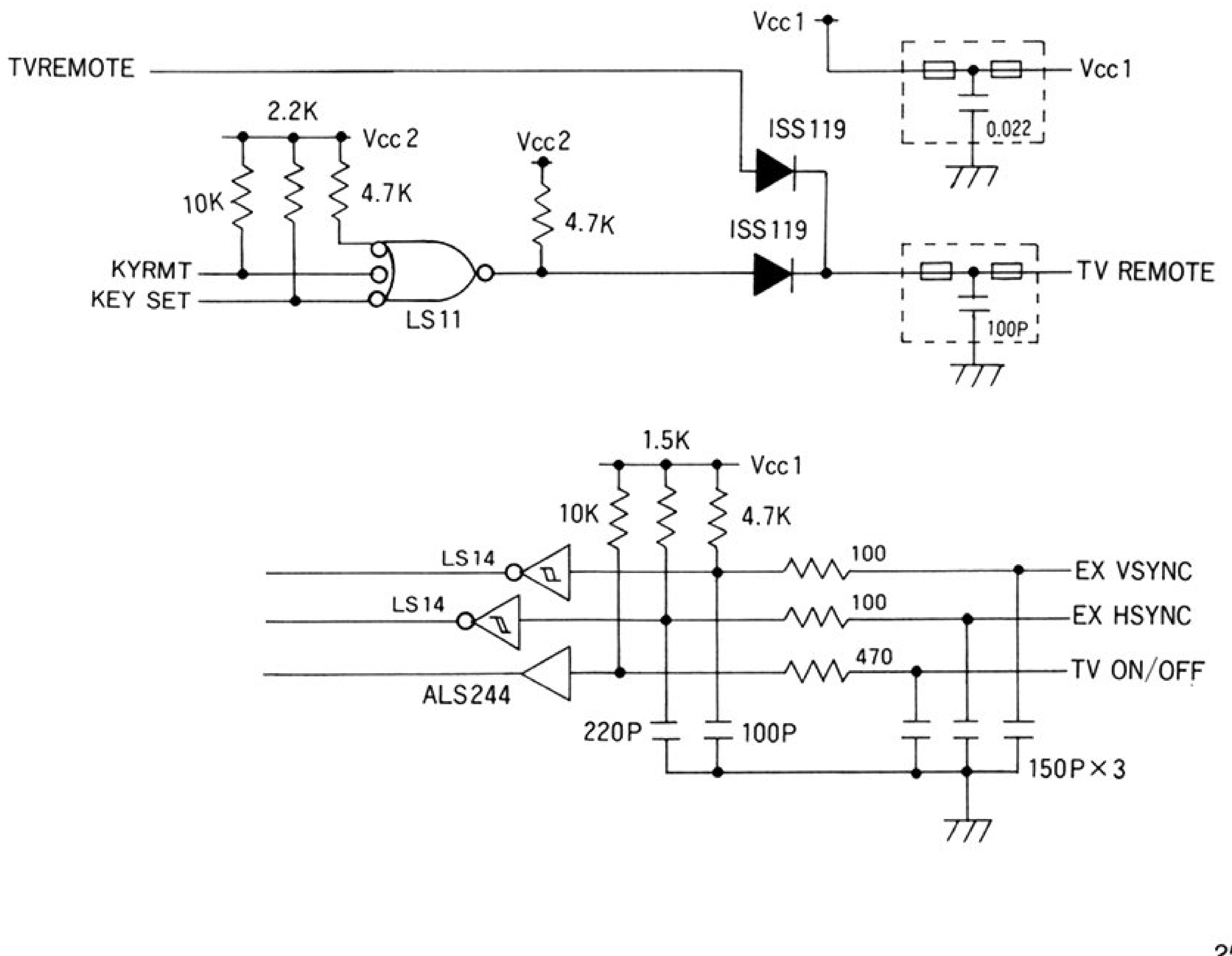
❷·3 ディスプレイ制御コネクタ

ディスプレイ制御コネクタはリモート制御信号のほか，ディスプレイ TVの電源 ON/OFF ステータスやディスプレイ側からの水平/垂直同期信号入力も配置されています。コネクタ周辺の回路は図5のようになっています。

リモート信号は，ディスプレイ TV 内部ではワイヤレスリモコン信号と同じものとして扱われているようで，信号波形もリモコンが出力しているものとほぼ同一となっています。X 68000では，基本的にリモコン出力はキーボードにまかせているのですが，キーボードが差し込まれていないときにも本体側でポートを操作すればリモコン出力が出せるようにしたり，キーボードからのリモート制御を禁止することができるようにしています。

本体から出力するリモート制御信号（TVREMOTE）と，キーボードからのリモート信号（KYRMT）は，それぞれダイオードを通して接続されています。KYRMT は KEYSET 信号によって抑えられるようになっており，KEYSET を'L'にしておけば，キーボードによるリモート制御はできなくなります。

●図……5　ディスプレイ制御コネクタ



TVREMOTEを'H'のままにすると，リモート制御信号は'H'のままになるため，やはりキーボードからの制御は利かなくなります。このときはワイヤレスリモコンによる制御もできなくなります（CZ-600 DEの場合）ので，ディスプレイTVの中ではこの信号とワイヤレスリモコンからの受信信号が単純に合成されているのではないかと思われます。

ディスプレイから送られてくる同期信号は，本体がスーパーインポーズ動作をするときに必要なものです。スーパーインポーズ動作では，TV画面とコンピュータ画面をきちんと同期させないと，どちらかの画面が流れてしまうことになります。TVの同期信号は放送局が送り出してくるものであり，タイミングの変更のしようがありませんから，TV放送の同期信号をディスプレイTVから受け取り，本体側の表示タイミングを合わせるようにしています。

❷・4 映像入力用コネクタ

オプションのカラーイメージユニットなどと接続して，画像取り込みを行うためのコネクタです。コネクタ近辺の回路は図6のようになっています。信号は5ビットのコマンドポート，クロックであるQA，D/A変換後のデータを渡すための16ビットのデータに分類されます。イメージユニットはQAに同期してD/A変換された映像データを本体に送るようになっており，本体側はこのデータをそのままグラフィックV-RAMに転送することで，画像取り込みが実現されています。

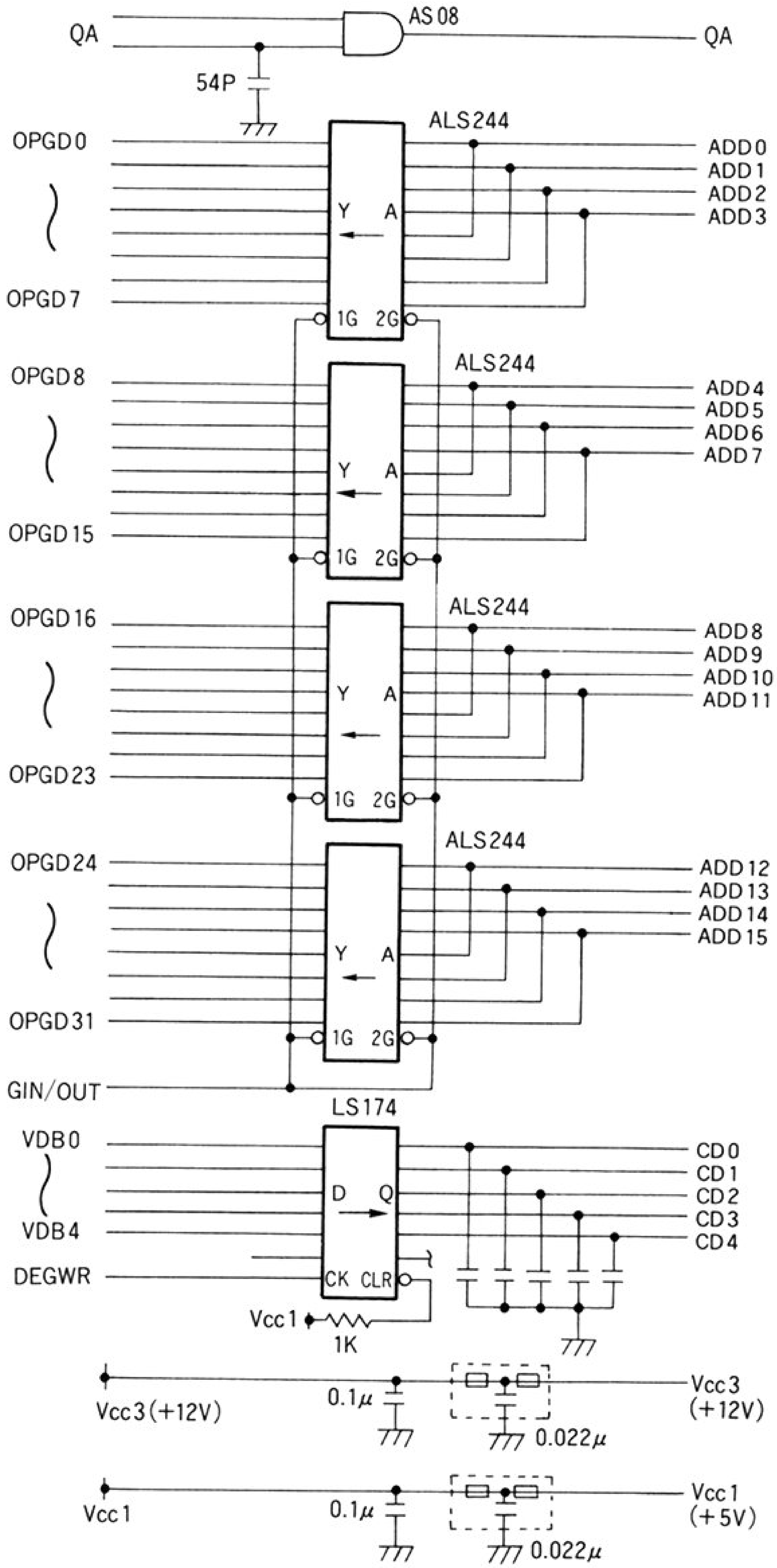
コネクタに出力されている＋12 V，＋5 Vの各電源ラインはフェライトビーズとコンデンサによるT型のフィルターを通してから引き出されており，ノイズが外部に漏れたり，外部で乗ってきたノイズが内部の電源ラインなどに悪影響を与えないようにしています。

❷・5 プリンタコネクタ

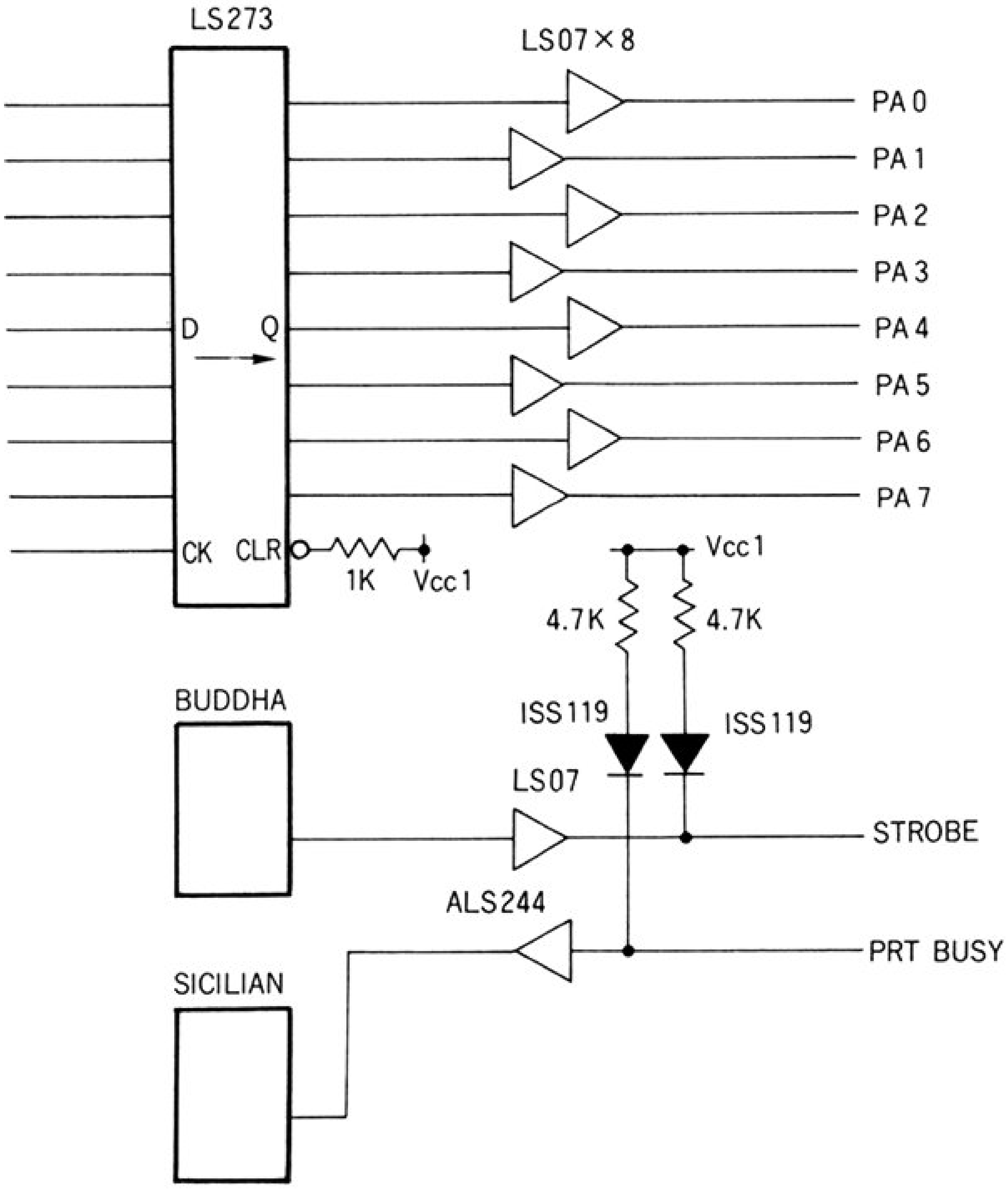
プリンタコネクタの周囲は図7のような回路となっています。X 68000のプリンタポートに用意されている信号は，セントロニクス準拠のプリンタとつなぐ上で最低限必要な，データであるSTROBE，BUSYのみになっており，PE（紙切れ検出）やONLINE（オンライン状態になっているか否か）はおろかACK（データ受け取り完了を示す信号）すらありません。このため，プリンタの異常はBUSYだけを見て判断するしかありません。

プリンタに送るデータはラッチ（LS 273）にセットされ，オープンコレクタバッファであるLS07を通ってプリンタと接続されます。プリンタ側にデータの受け取りを指示するSTROBE信号はゲートアレイの1つ，BUDDHAから出力され，LS 07でバッファされてプ

●図……6　映像入力用コネクタ



●図……7　プリンタコネクタ



リンタと接続されます。電圧レベルを正しく出すため，4.7 KΩ でプルアップするとともに，ダイオードを入れることで，本体の電源が切れているときやプリンタ側の出力電圧が本体よりも高い場合，プリンタから本体内部に電流が逆流することのないようにしています。

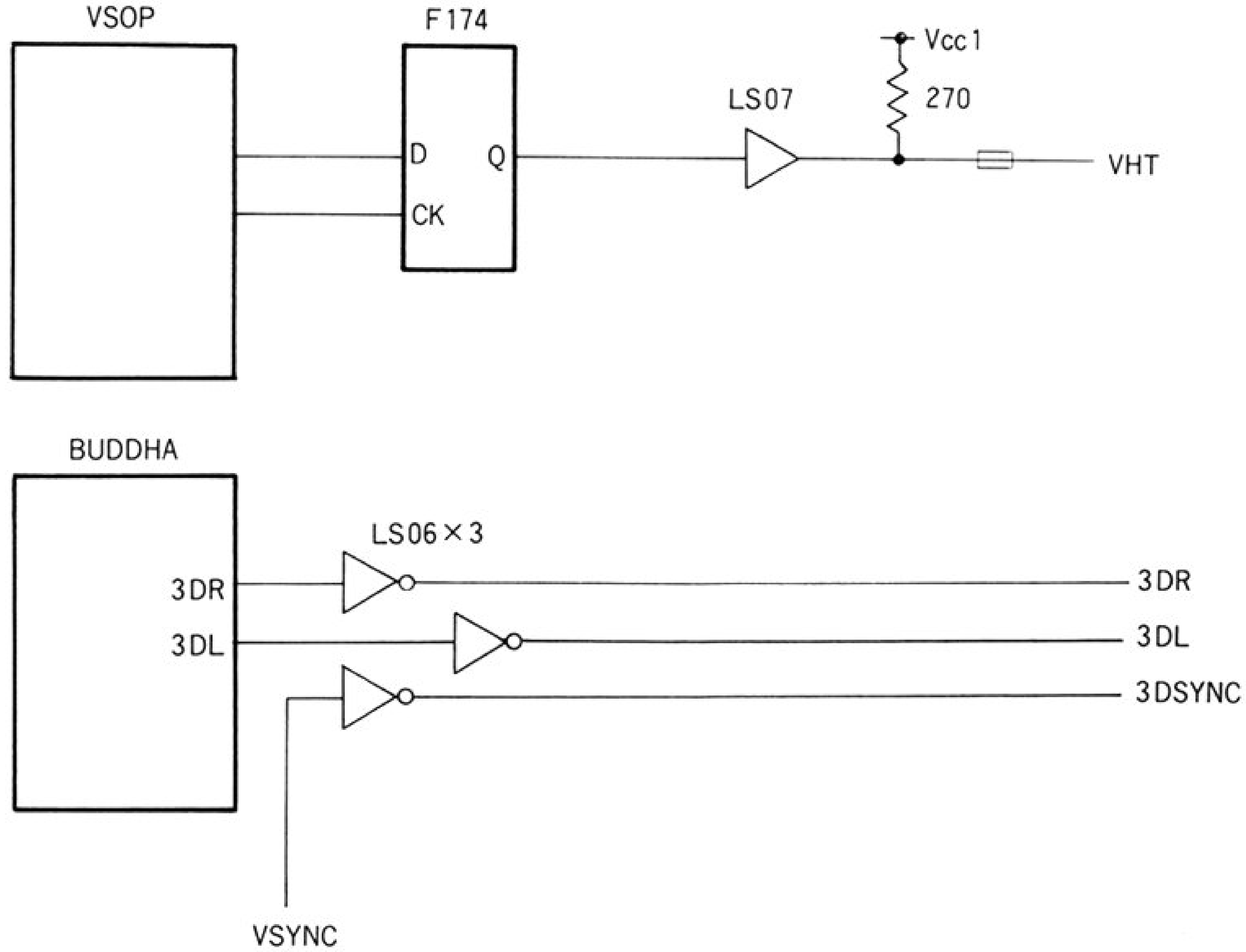
BUSY (図7ではPRT BUSY) はプリンタがデータを受け取れる状態にあるか否かを示すものです。入力ピンであるため，プリンタがつながらないときにピンがハイ・インピーダンス状態になって誤動作することのないようプルアップが行われ，ケーブルが差し込まれていないときはBUSYのままになるようにしています。プルアップはSTROBE信号と同様の方法がとられています。

❷·6 シースルーカラー端子/3Dスコープ端子

シースルーカラー端子，3Dスコープ端子まわりの回路を図8に示します。シースルーカラー端子は半透明領域を示すもの，3Dスコープ端子は液晶シャッターを付けたときの各シャッターの開閉信号と，垂直同期信号が配置されています。

各信号ともオープンコレクタ出力となっていますが，シースルーカラー端子は270Ωのプルアップがされており，さらにフェライトビーズを通すことで不要輻射を減らすようにしています。

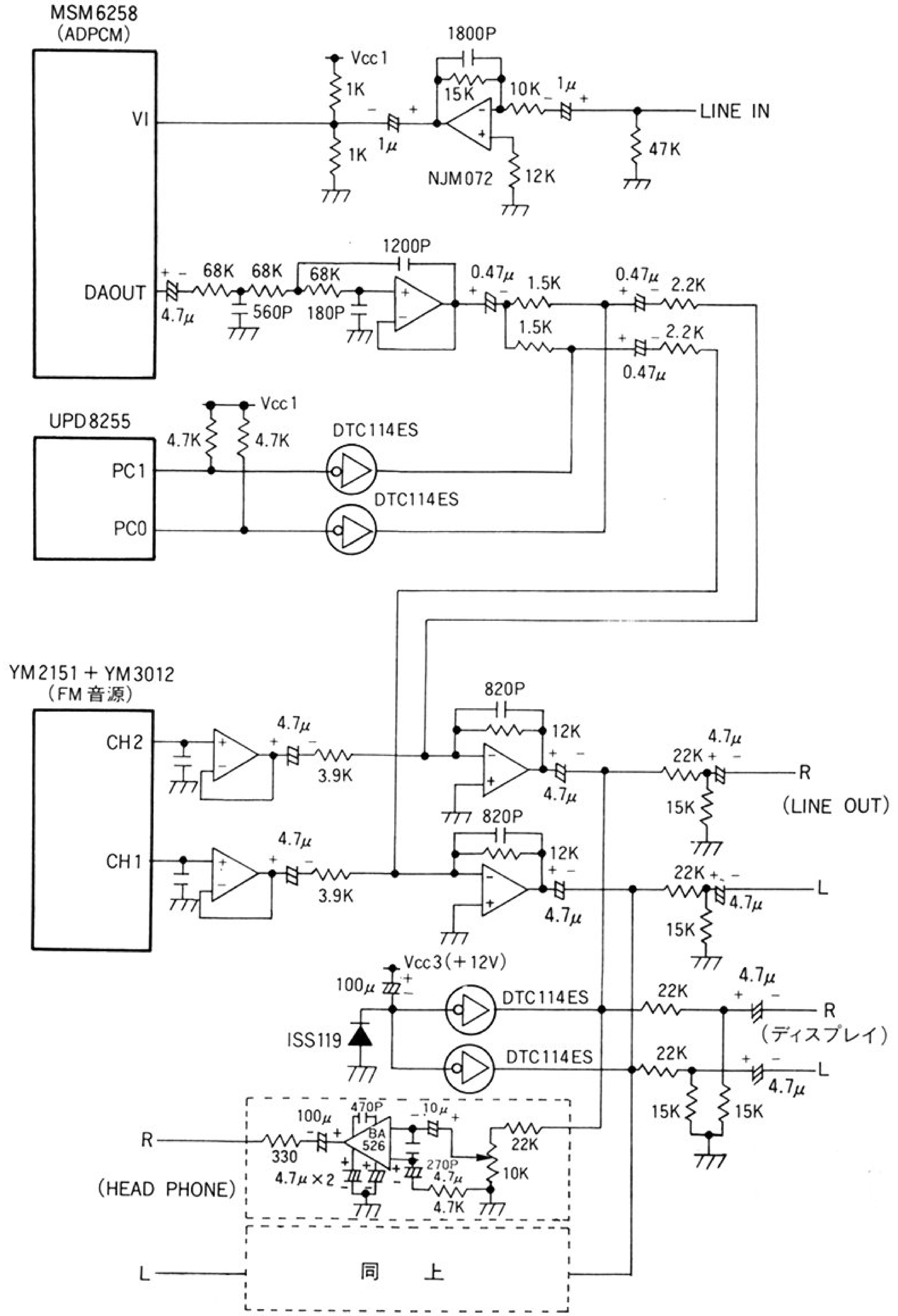
●図……8　シースルーカラー端子/3Dスコープ端子



❷·7 ライン入出力/ヘッドフォン関係

ライン入出力やヘッドフォンなどの，オーディオ関係の回路は図9のようになっています。周波数特性の保証やDC成分カット用のコンデンサなどが多く，やや混雑しているようですが，一つ一つ順に見ていけばさほどややこしいものではありません。

●図…… 9　オーディオ関係


MSM6258
(ADPCM)
VI
DAOUT
Vcc1
1K
1K
1μ
1800P
15K
10K
1μ
NJM072
12K
LINE IN
47K
4.7μ
68K
68K
68K
560P
180P
1200P
0.47μ
1.5K
1.5K
0.47μ
2.2K
2.2K
0.47μ
UPD8255
PC1
PC0
4.7K
4.7K
Vcc1
DTC114ES
DTC114ES
YM2151 + YM3012
(FM音源)
CH2
CH1
4.7μ
3.9K
4.7μ
3.9K
820P
12K
820P
12K
4.7μ
4.7μ
22K
4.7μ
R
(LINE OUT)
15K
22K
4.7μ
L
15K
Vcc3(+12V)
100μ
ISS119
DTC114ES
DTC114ES
22K
22K
4.7μ
R
(ディスプレイ)
4.7μ
L
15K
15K
R
(HEAD PHONE)
330
100μ
470P
BA526
10μ
270P
4.7μ×2
4.7μ
4.7K
22K
10K
L
同　上

ライン入力はNJM 072によって反転増幅されたのち，1 μFのコンデンサでDC分をカットされ，2.5 Vを中心とする信号波形にされたのちADPCMチップ(MSM6258)の入力端子に入ります。反転増幅回路では1800 pFのコンデンサによるハイカット特性を持たせており，ADPCMの動作に悪影響を及ぼす高い周波数成分を減衰させるようにしています。

音声出力の方はFM音源やパンポット制御などが絡むので少々複雑です。ADPCMから出力された音声出力は，階段状の波形をならすためにローパスフィルター兼バッファアンプを通った後，左右均等に分割されます。この直後にあるDTC114ESはパンポット回路です。DTC 114 ESによって任意の側の出力を強制的に0 Vにしてしまうことで，ADPCMから出力されてきた音声信号をカットしてしまうわけです。それぞれのDTC114ESはμPD 8255のPC 0とPC 1で制御されており，出力が'H'のときにDTC114ESの出力が'L'になり，音声出力がOFFとなります。

このようにして得られた音声出力はバッファアンプを通ったFM音源の出力と合成された後，反転増幅されます。この反転増幅回路の出力はラインアウト，ディスプレイ制御コネクタ，ヘッドフォン/内部スピーカ用に分割されます。

ヘッドフォン/内部スピーカ用に分離された出力は音量調整された後，BA 526によるオーディオアンプで増幅され，ヘッドフォンやスピーカをドライブします。

❷・8 ジョイスティックコネクタ

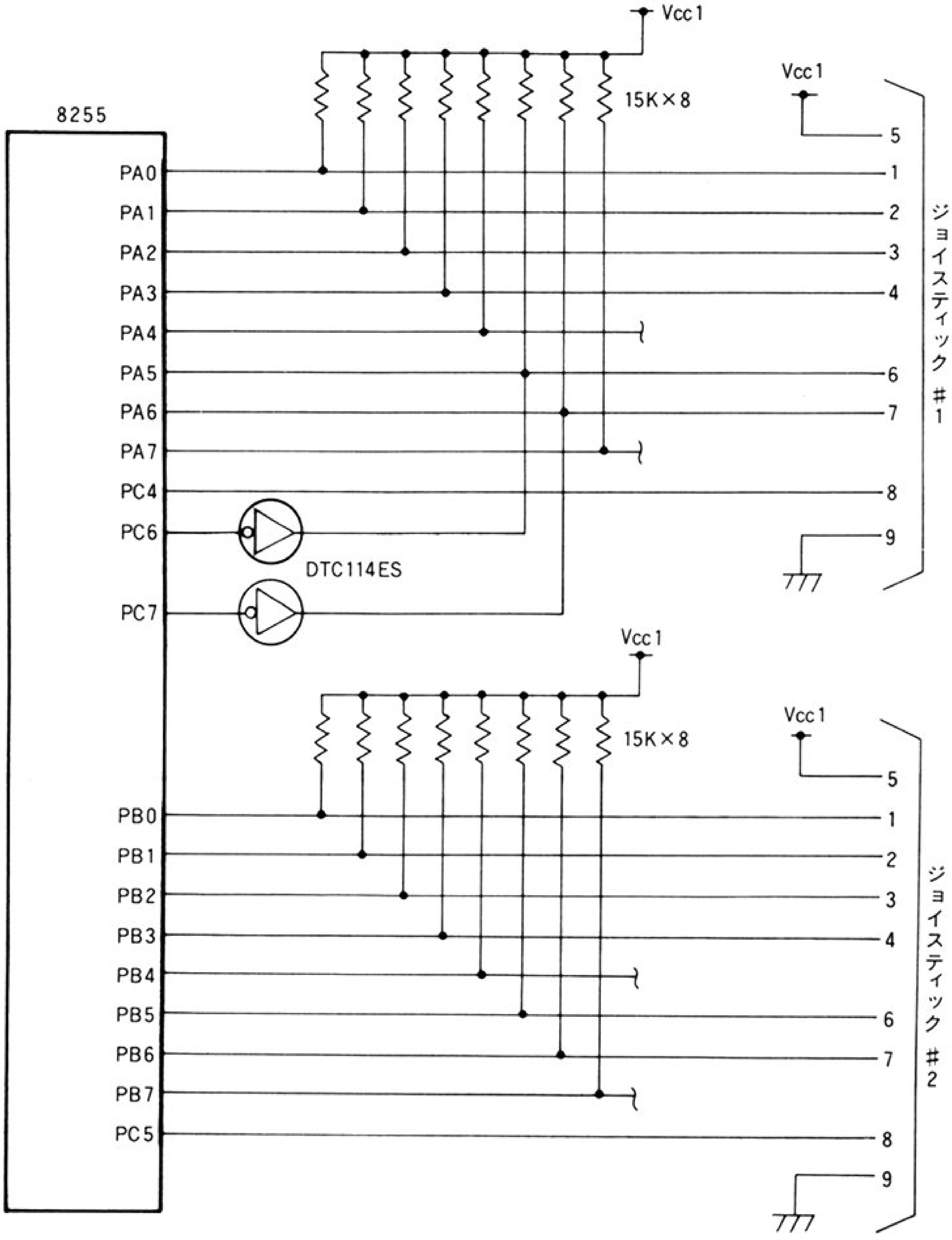
ジョイスティックコネクタまわりの回路を図10に示します。ジョイスティック#2はPROタイプ以外では背面にあります。

ジョイスティック#1と#2はいずれも8255で制御されます。データはそれぞれポートAとポートBで読み出され，制御はPCで行われます。

少し異なるのはジョイスティック#1の方は6番ピンと7番ピンが出力ピンとしても使えるようになっていることです。PC 6とPC 7を使ってポートAの動作モードを変更しなくても出力ピンとして使うことができるようにしているわけです。

入力として使われるポートAやポートBはすべて15 KΩ抵抗でプルアップされていますが，ポートCはそのままコネクタに出されています。ポートCを入力ポートとしてプログラムすると，これらのピンが浮き上がってしまいますので，避けた方がよいでしょう。また，リセット直後，8255はすべてのポートを入力ピンとするようになっています。IPL-ROMを自分で書き換えるようなときは，リセット後，できるだけ早い時期にモード設定を行うようにすべきでしょう。

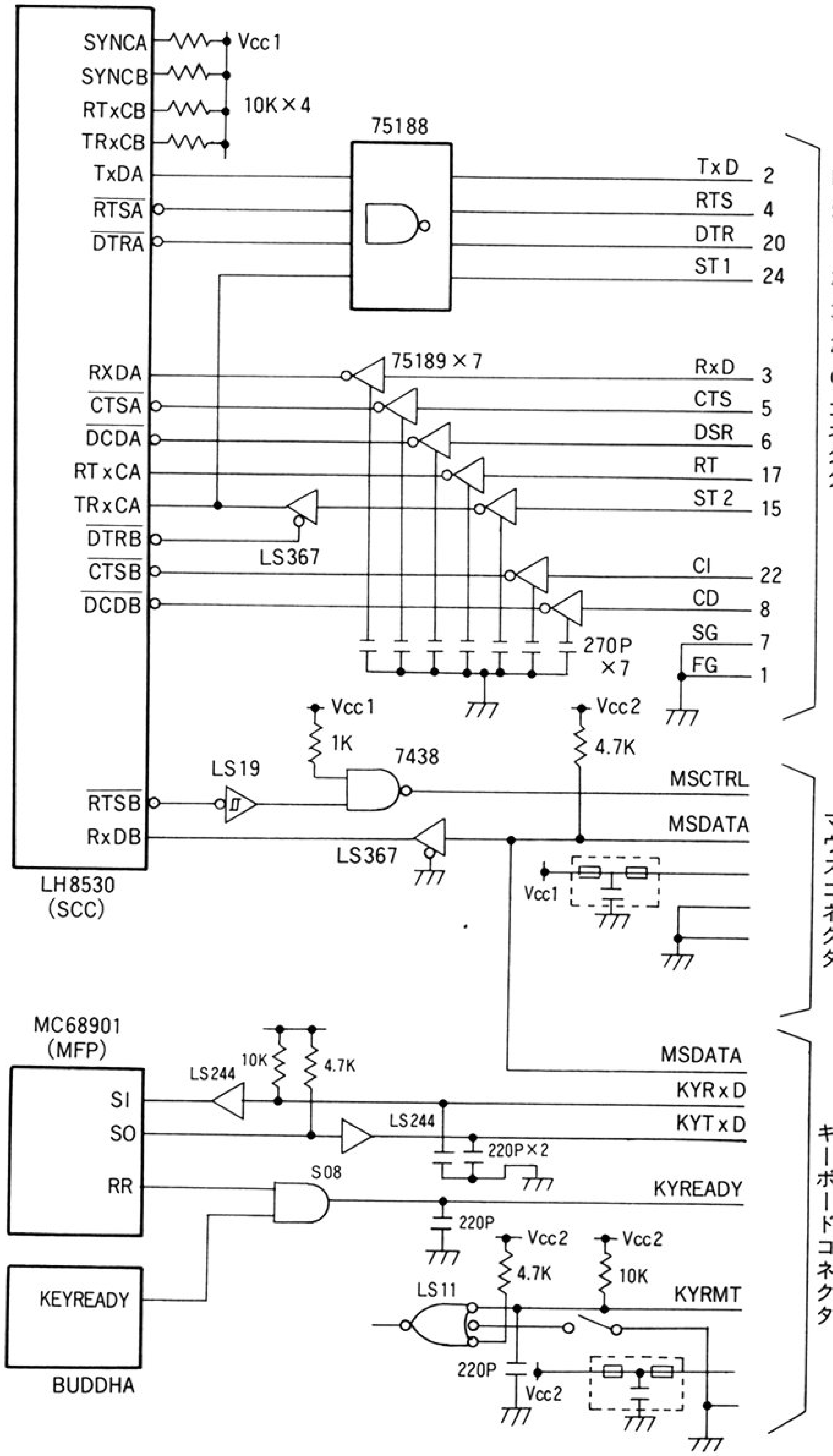
●図……10　ジョイスティックコネクタ



❷·9　RS-232C/マウス/キーボード

RS-232C，マウス，キーボードの各コネクタまわりの回路を図11に示します。X68000ではマウスやキーボードもシリアル伝送で行っており，またキーボードにマウスコネクタがあるなど，互いに関連しあっていますので，ここでは1つの図にまとめてみました。RS-232Cとマ

●図……11 RS-232C/マウス/キーボードコネクタ



ウスデータの入力はシリアル伝送用 LSI(LH 8530)で，キーボードとのシリアル伝送は MFP (68901) に付いているシリアルポートを利用して行っています。

まず，LH 8530 の方から見ていきます。基本的にはチャンネルAが RS-232 C用，チャンネルBがマウス用となっていますが，LH 8530 は RS-232 C のサポートには少々信号が足りなく，またマウス用にはコントロール信号が1つと受信データピンだけがあればよいということから，チャンネルBの余った信号ピンを RS-232 C 用に振り向けるなど，信号の入れ替えが行われています。

チャンネルAの DCD ピンは 232 C コネクタでは DSR, チャンネルBの CTS, DCD はそれぞれ CI, CD に，DTR は TRxCA（送信クロック）を外部から取り入れるバッファの ON/OFF をするための信号として使用しています。

MSCTRL（マウス制御信号）は，マウスからのデータ転送開始を指示するための信号，MSTDATA（マウスデータ信号）はマウスの移動量やボタンの状態を送ってくる信号線です。RS-232 C の信号レベルは＋12 V～－12 V ですが，マウスの信号はいずれも TTL レベルの信号となっています。マウスデータは，マウスコネクタのほか，キーボード側からくることもある（キーボードにもマウスコネクタがあります）ことから，キーボード側とマウス側の信号が接続されていますが，これは AND ゲートなどで合成されるのではなく，単純に結線されているだけです。マウスデータは LS 367 でバッファされたのち，LH 8530 のチャンネルBの RxD 端子に接続されています。

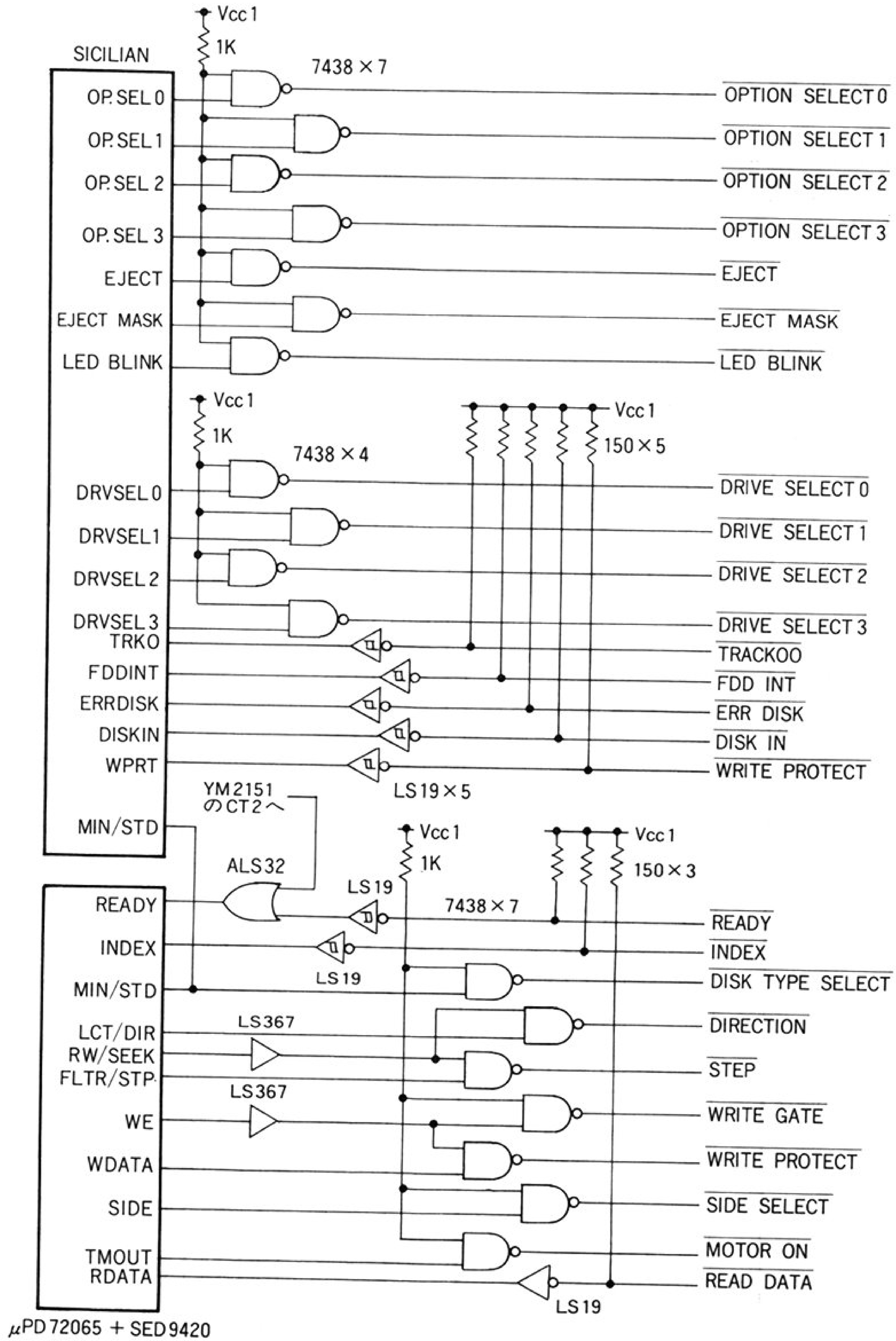
キーボードのデータ入出力や制御信号などもマウスと同様，TTL レベルの信号です。通常，キーボードは押されたり離されたりしたキーの番号を本体に送るだけですから，制御信号とデータ信号があればよさそうなものですが，X 68000 の場合にはキーボードにマウスコネクタがあるほか，キーボードの LED の点滅や明暗制御，ディスプレイ TV 制御（チャンネル切り替え，ボリューム変更，モード変更など）も行います。このため，コネクタの信号配置もかなりにぎやかになっています。

マウスコネクタ，キーボードコネクタとも電源ピンを持っています。いずれも外部からのノイズの侵入や内部ノイズの放出を避けるため，フェライトビーズとコンデンサを組み合わせたT型のフィルターが組まれています。

❷·10 外部FDDコネクタ

外部 FDD コネクタの周辺回路を図 12 に示します。X 68000 は FDC（フロッピーディスクコントローラ）にμPD765を，データセパレータとしてSED9420を使用しています。通常のFDDであればこれらのLSIにバッファを付ける程度で接続できるのですが，X68000の場合

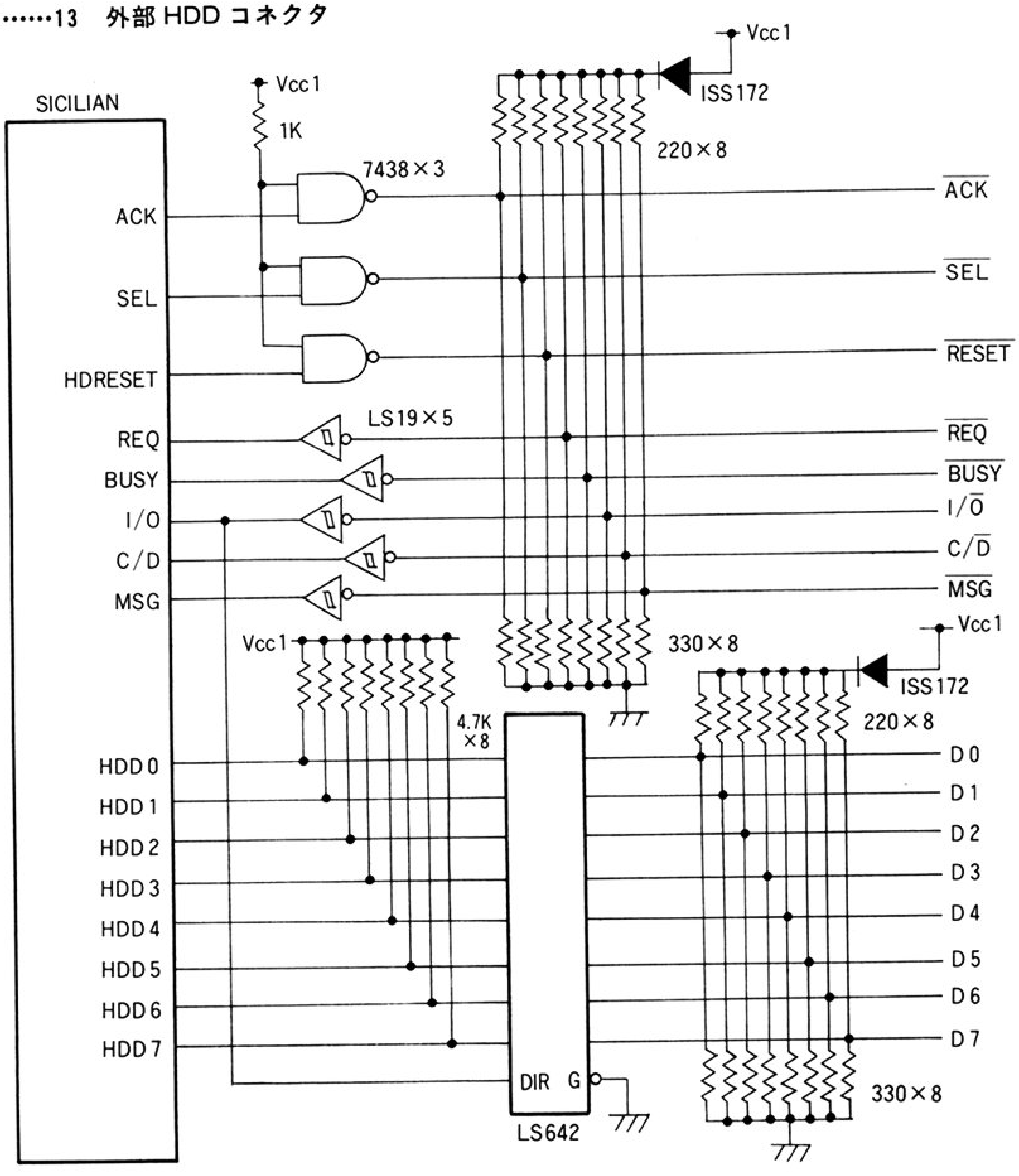
●図……12　外部FDDコネクタ



にはLEDの点滅やオートエジェクトなどの特殊機能があるため，専用ゲートアレイSICILIANで信号の補完を行っています。回路を見ると，SICILIANからの制御信号で2HDと2DDの切り替えをサポートしようとしているようなのですが，実際には2HDしか扱えないというのが残念なところです。

❷·11 外部HDDコネクタ

●図……13　外部HDDコネクタ



外部HDDコネクタ周辺回路を図13に示します。X 68000の元祖タイプ以来SUPERまでHDDインタフェースはSASIバスのものが使われていました。伝送制御はI/Oコントロール用のゲートアレイ，SICILIANがすべて受け持っています。

データ入出力信号はLS 642が使用され，制御系の出力は電流引き込み能力の大きいオープンコレクタバッファである7438が，入力側はPNP入力のシュミットトリガゲートである，LS 19が使われています。

信号線はすべて220Ωと330Ωのプルアップ/プルダウン抵抗で終端されており，信号の反射による波形の乱れを抑えるようになっています。

3 ハードウェアの変遷

X 68000は初代機以来，その基本的なアーキテクチャをまったく変えることなく機種を追加してきています。機能ブロックレベルではSUPER以降でHDDインタフェースがSASIからSCSIに変更された以外はほとんど変更がないくらいですが，少し細かく見ると，微妙に変更されている部分を見つけることができます。

ここでは機種の変更とともに変わってきた主な部分を整理しておくことにします。

3·1 ゲートアレイの変遷

表1～表9に各機種ごとの，使われている主要ゲートアレイの一覧を示します。これをもう少し変更箇所がわかりやすいようにまとめたのが表10です。

ゲートアレイの構成は初代機からACEへの移行時にもっとも大きく変わりました。初代機の9つのゲートアレイはACE時に全面的に見直しが行われ，7つに集約されました。初代機と共通で使われているのはスプライトコントローラのCYNTHIAだけです。CYNTHIAは初代機の時点ですでに完成したものとみられ，全機種でそのまま使われ続けています。

ACEからEXPERTへの移行ではメモリコントローラとI/Oコントローラに変更が行われ，さらにSUPERではHDDインタフェースがSASIからSCSIになったためI/Oコントローラが変更され，SASIインタフェースピンの廃止，SCSIコントローラのサポートがなされました。

●表……1　主要ゲートアレイ一覧　X 68000（CZ-600 CE）

機　能	型　名	愛　称
メモリコントローラ	IX0904CE	ET
システムコントローラ	IX0905CE	BUDDHA
スプライトコントローラ	IX0906CE IX0908CE	CYNTHIA CYNTHIA JR
CRTコントローラ	IX0902CE IX0903CE	VINAS 1 VINAS 2
ビデオコントローラ	IX0901CE	VSOP
ビデオデータセレクタ	IX0907CE	RESERVE
I/Oコントローラ	IX0909CE	SICILIAN

●表……2　主要ゲートアレイ一覧　X 68000 ACE（CZ-601 C/611 C）

機　能	型　名	愛　称
メモリコントローラ	IX1097CE	OHM
システムコントローラ	IX1099CE	MESSIAH
スプライトコントローラ	IX0906CE	CYNTHIA
CRTコントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/Oコントローラ	IX1098CE	IOSC

●表……3　主要ゲートアレイ一覧　X 68000 EXPERT（CZ-602 C/612 C）

機　能	型　名	愛　称
メモリコントローラ	IX1197CE	OHM2
システムコントローラ	IX1099CE	MESSIAH
スプライトコントローラ	IX0906CE	CYNTHIA
CRTコントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/Oコントローラ	IX1264CE	IOSC-2

●表……4　主要ゲートアレイ一覧　X 68000 PRO（CZ-652 C/662 C）

機　能	型　名	愛　称
メモリコントローラ	IX1266CE	McCOY
システムコントローラ	IX1267CE	SCOTCH
スプライトコントローラ	IX0906CE	CYNTHIA
CRTコントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/Oコントローラ	IX1264CE	IOSC-2

●表……5　主要ゲートアレイ一覧　X 68000 EXPERT 2（CZ-603 C/613 C）

機　能	型　名	愛　称
メモリコントローラ	IX1197CE	OHM2
システムコントローラ	IX1099CE	MESSIAH
スプライトコントローラ	IX0906CE	CYNTHIA
CRTコントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/Oコントローラ	IX1264CE	IOSC-2

●表……6　主要ゲートアレイ一覧　X 68000 PRO 2（CZ-653 C/663 C）

機　能	型　名	愛　称
メモリコントローラ	IX1266CE	McCOY
システムコントローラ	IX1267CE	SCOTCH
スプライトコントローラ	IX0906CE	CYNTHIA
CRTコントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/Oコントローラ	IX1264CE	IOSC-2

●表……7　主要ゲートアレイ一覧　X 68000 SUPER-HD（CZ-623 C）

機　能	型　名	愛　称
メモリコントローラ	IX1197CE	OHM2
システムコントローラ	IX1099CE	MESSIAH
スプライトコントローラ	IX0906CE	CYNTHIA
CRT コントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/O コントローラ	IX1604CE	PEDEC

●表……8　主要ゲートアレイ一覧　X 68000 SUPER（CZ-604 C）

機　能	型　名	愛　称
メモリコントローラ	IX1197CE	OHM2
システムコントローラ	IX1099CE	MESSIAH
スプライトコントローラ	IX0906CE	CYNTHIA
CRT コントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/O コントローラ	IX1604CE	PEDEC

●表……9　主要ゲートアレイ一覧　X 68000 XVI（CZ-634 C/644 C）

機　能	型　名	愛　称
メモリコントローラ	IX1748CE	ASA
システムコントローラ	IX1749CE	DOSA
スプライトコントローラ	IX0906CE	CYNTHIA
CRT コントローラ	IX1093CE	VICON
ビデオコントローラ	IX1095CE	VIPS
ビデオデータセレクタ	IX1094CE	CATHY
I/O コントローラ	IX1604CE	PEDEC

●表……10　主要ゲートアレイの変遷

<table>
<tr><th></th><th>X68000</th><th>ACE</th><th>EXPERT</th><th>EXPERT2</th><th>SUPER</th><th>XVI</th><th>PRO</th><th>PRO2</th></tr>
<tr><td>メモリコントローラ</td><td>ET</td><td>OHM</td><td colspan="4">OHM2</td><td colspan="2">McCOY</td></tr>
<tr><td>システムコントローラ</td><td>BUDDHA</td><td colspan="4">MESSIAH</td><td>DOSA</td><td colspan="2">SCOTCH</td></tr>
<tr><td>スプライトコントローラ</td><td>CYNTHIA
CYNTHIA JR</td><td colspan="7">CYNTHIA</td></tr>
<tr><td>CRTコントローラ</td><td>VINAS 1
VINAS 2</td><td colspan="7">VICON</td></tr>
<tr><td>ビデオコントローラ</td><td>VSOP</td><td colspan="7">VIPS</td></tr>
<tr><td>ビデオデータセレクタ</td><td>RESERVE</td><td colspan="7">CATHY</td></tr>
<tr><td>I/Oコントローラ</td><td>SICILIAN</td><td>IOSC</td><td colspan="2">IOSC-2</td><td colspan="2">PEDEC</td><td colspan="2">IOSC-2</td></tr>
</table>

PROの系列は他の縦型の機種とは少し異なっており，メモリコントローラ，システムコントローラが独自のものとなっています。機能的には特に変更する意味があるとは思えませんので，おそらく低価格なパッケージのものに作り替えたのではないかと思われます。

❸·2　その他の主な変更点

X 68000の各機種における主な変更点のうちゲートアレイ以外のものを表11にまとめてみましたので参考にしてください。初代機とACE以降ではFDDユニットが異なっていたり，ACE時にHDD内蔵型と非内蔵型で電源を区別（トランスや＋5 Vの平滑コンデンサの容量に違いがあります）したのをEXPERT以降では一本化するなど，部品を流用する上で注意が必要そうなところがあります。電源については，ACEにHDDを内蔵したときに電源容量が不足したためトランスを巻き直して対応し，EXPERT以降では最初からHDD内蔵を考慮した電源設計となったため，共通化されたのではないかと思われます。

その他の変更はIPL-ROMの内容変更や拡張ROM用のICソケットの変更など，ソフトウェアの変更やデバイスの容量増加に伴うものがほとんどです。

●表……11　ゲートアレイ以外の主な変更点

機　種　変　更	主　な　変　更　内　容
元祖からACE	○FDDユニットの変更　(DUNTK5716DE01) ・逆転エジェクトスイッチ廃止 (初代機のFDDユニットは使用不可) ○電源ユニットの変更 ・UADP-0058CEZZ(ACE用)/UADP-0057CEZZ(ACE-HD用)
ACEからEXPERT	○電源ユニットの変更　(UADP-0069CEZZ) ○HDDユニットの変更　(DMECH0002CE01:40Mバイト) ○"HIGH RESO"LEDの廃止
EXPERT2からSUPER	○HDDユニットの変更　(DMECH0003CE01:80Mバイト) ○4Mビットマスク ROM変更　(IX1614CE/IX1615CE) ○拡張用ICソケットを1Mマスク ROM対応に ○SCSIコントローラ追加，SASIインターフェース廃止
SUPERからXVI	○メインメモリ増設用コネクタ追加 ○CPUの変更　(HD68HC000PS10からHD68HC000B16) ○FPUソケット追加 ○MPUクロックを16.67MHz/10MHzの2通り選択可能に ○4Mビットマスク ROM変更　(IX1775CE/IX1776CE) ○BIOS ROM切り替えICソケットを増設RAMボード上に移動

拡張スロット仕様

高速のデータ転送を要求したり，CPUと密着した動作を行うようなオプションボードを接続するポートとしては拡張スロットに勝るものはありません。この章ではX68000の拡張スロットの電気的，機械的な規約について説明します。

1 はじめに

X 68000のオプションボードを自作したり，市販のボードの回路図を読むときに必須となってくるのが拡張スロットの仕様です。市販のボードの回路図をまねしたり，一部を流用するといった方法でオリジナルのボードを作ることもある程度は可能ですが，市販ボードが使っていないようなLSIを接続したり，タイミングに制約があるような回路を接続するような場合には，スロットに許されている最大消費電流や動作タイミングなどをおさえて設計しないと，一見まともに動いているように見えてもしばらく動かしていると突然変な動作をするようになったり，本体の動作に影響が出るといった，解析のむずかしい現象に悩まされることになりかねません。

この章では，X 68000の拡張ボードの機械的な仕様（寸法），拡張スロットのDC規格（消費電流の制限），スロットの各信号の意味やそれぞれの動作タイミングなど，拡張スロットを使用する上で必要となる情報を整理しています。

X 68000の拡張スロットは単なるCPUによるリード/ライトだけではなく，割り込み，DMA，D-RAMリフレッシュ，バス交換などの動作がありますのでやや複雑に感じるかもし

れませんが，１つずつ分けて見ていけばさほど複雑なものではないでしょう。

●2 拡張ボード基板，パネル外形図

拡張ボードを個人レベルで作るときには拡張ボード用のユニバーサル基板を使い，パネルなどは付けないのが普通ですが，ある程度まとまった数の基板を作るときには基板製造を専門に行っているところに依頼した方が早く確実なものが作れます。このようなときには基板の各部の正確な寸法が要求されることになります。また，自作の回路が果たして基板上に収まるか，パネル面に部品が付くかなどを検討する場合も，寸法図があった方が便利です。ここではシャープから推奨されている X 68000 用拡張ボードの基板と I/O スロットカバー（パネル）の寸法を載せておきます。部品配置の検討やボード製作時の参考にしてください。

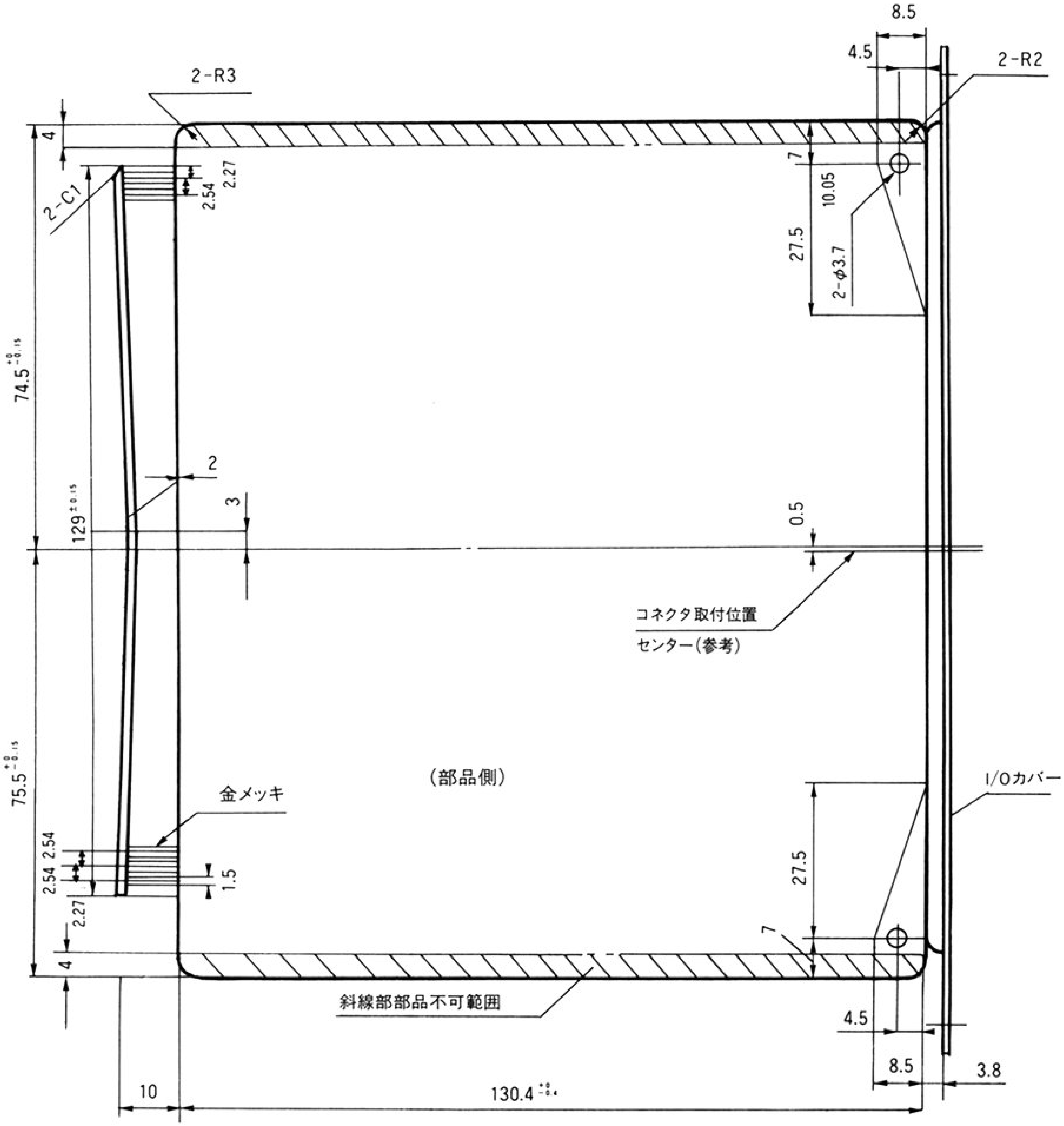
❷·1 拡張ボード外形

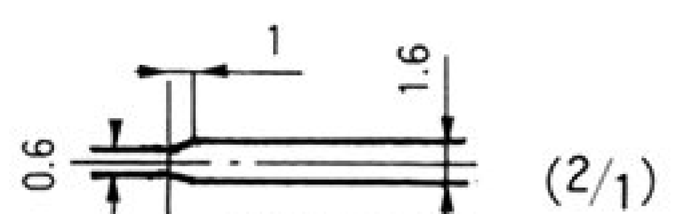
拡張ボードの推奨基板外形を図１に示します。図の面が部品実装面になっています。拡張ボードの基板厚は 1.6 mm と，ごく一般的な値です。上下エッジから 4 mm はガイドレールとの衝突を防ぐため，部品実装禁止になっています。パネル側の２つの穴と三角形の部品実装禁止領域はカードエジェクタを使うときのためのものです。

❷·2 パネル外形

X 68000 のパネルは通常の幅の狭いタイプのものと，PRO タイプのスロット＃4 や拡張 I/O ボックス用の幅の広いタイプの２種類があります。通常のスロットのパネル外形を図２に，PRO タイプのスロット＃4 や拡張 I/O ボックス用のパネル外形を図３に示します。X 68000，特にマンハッタンシェイプ型は非常にコンパクトにできているためスロット間隔が狭く，パネル面がやや窮屈になっています。スロット＃4 や拡張 I/O ボックスではこのあたりを考慮してパネル面を広くとれるようにしたのでしょう。

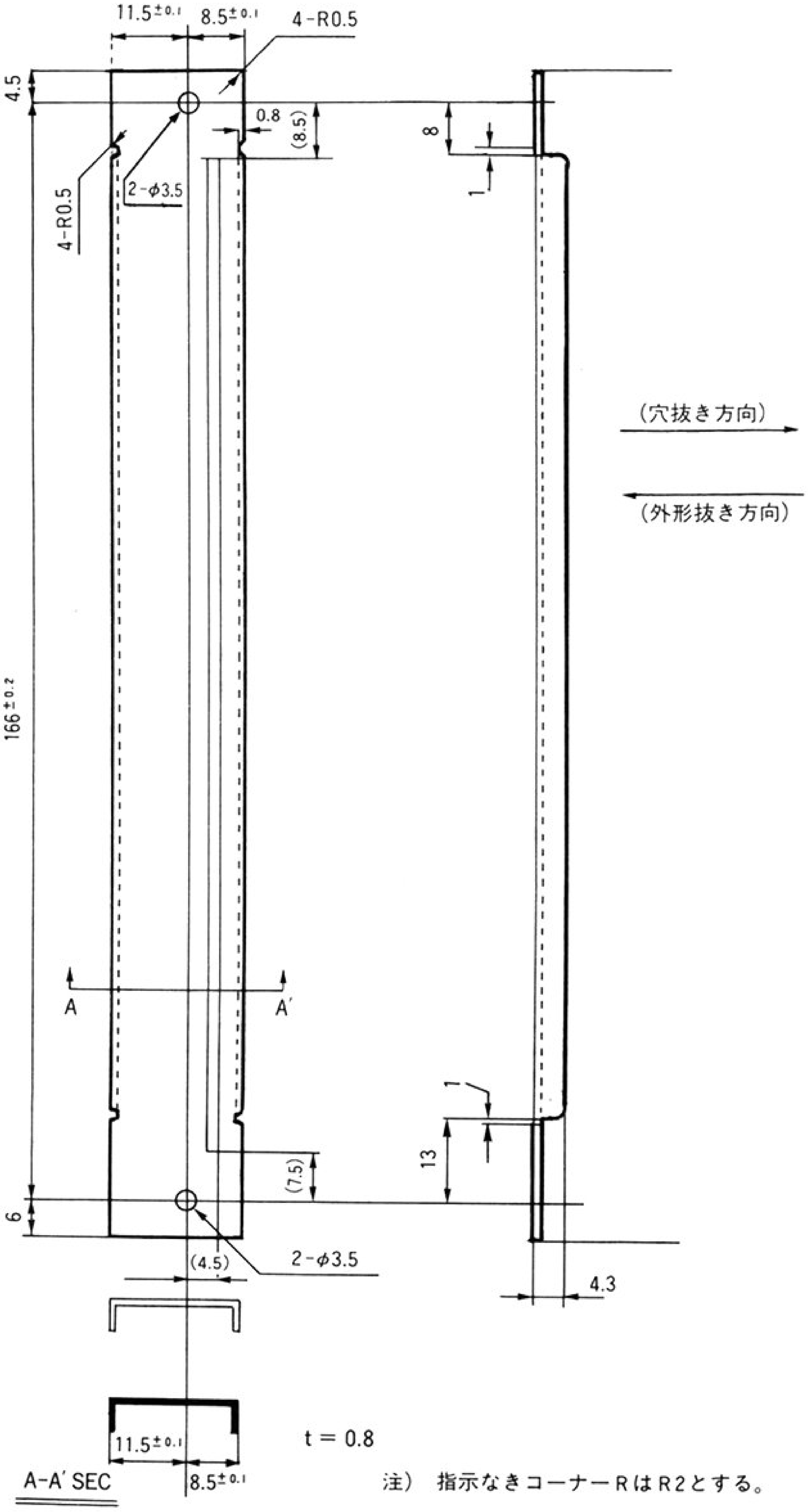
●図……1　I/O スロット基板外形寸法図



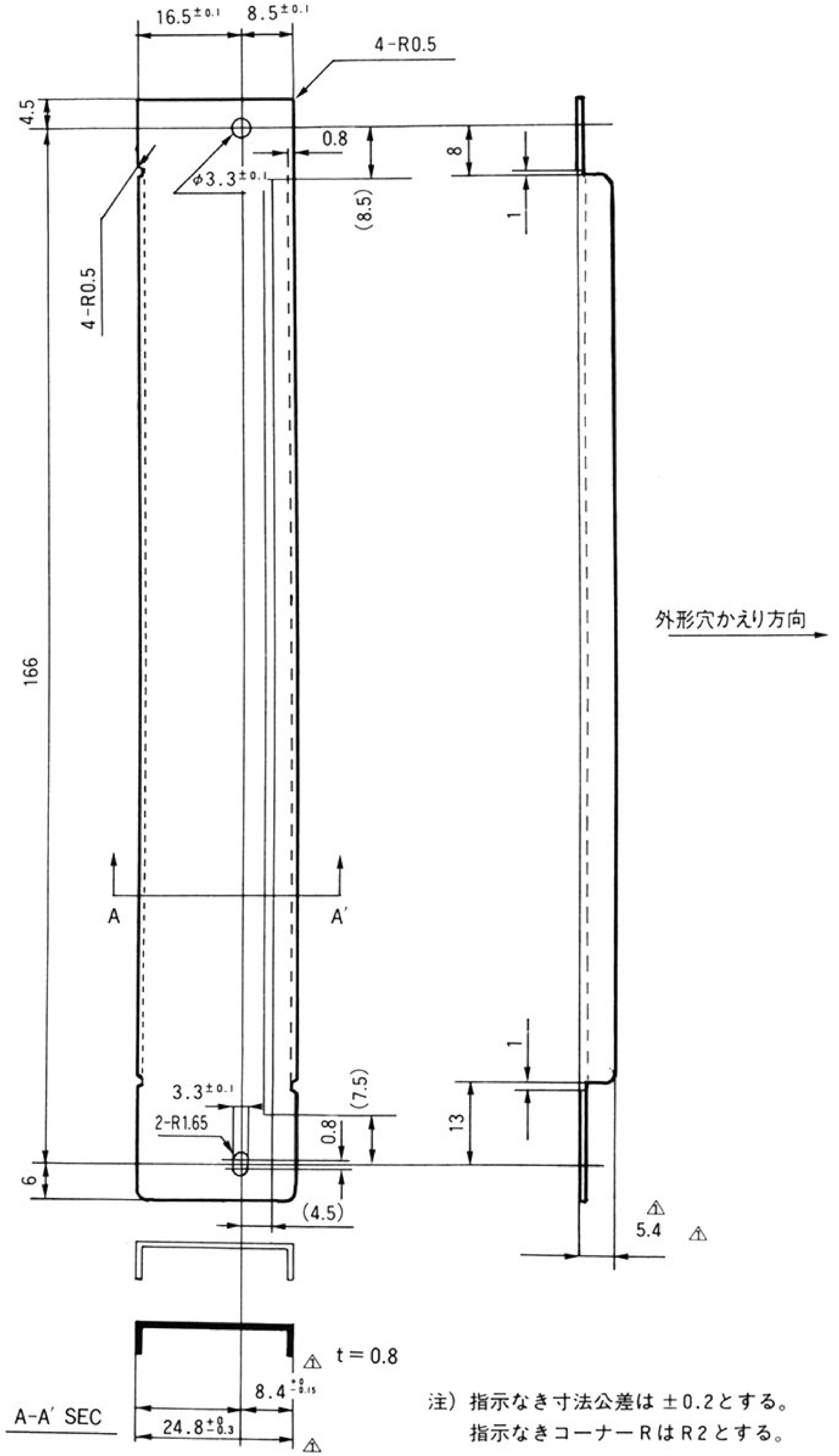


注)　部品の高さは，基板表面よりMAX13mmとする。部品脚の長さは基板表面よりMAX3.5mmとする。

●図……2　I/O スロットカバー（本体スロット#1～#3）



●図……3　I/O スロットカバー（本体スロット#4，拡張 I/O ボックス）



注）指示なき寸法公差は ±0.2とする。
　　指示なきコーナーRはR2とする。

●3 I/Oスロットの信号

ここでは，拡張スロットの各信号について説明します。拡張スロットの各信号はすべてTTLレベルとなっています。各信号の後ろに示したトーテムポールやオープンコレクタ，トライステートという名称はその信号が出力の場合，本体側のドライバICの種別*1を，入力の場合にはオプションボード側でその信号をドライブする方法について示しています。オープンコレクタと示されている入力信号をオプションボード側でドライブする場合，ドライバICにはオープンコレクタ，またはトライステートのバッファを用い，信号をドライブする必要のない場合は出力をハイ・インピーダンス状態にしておきます。

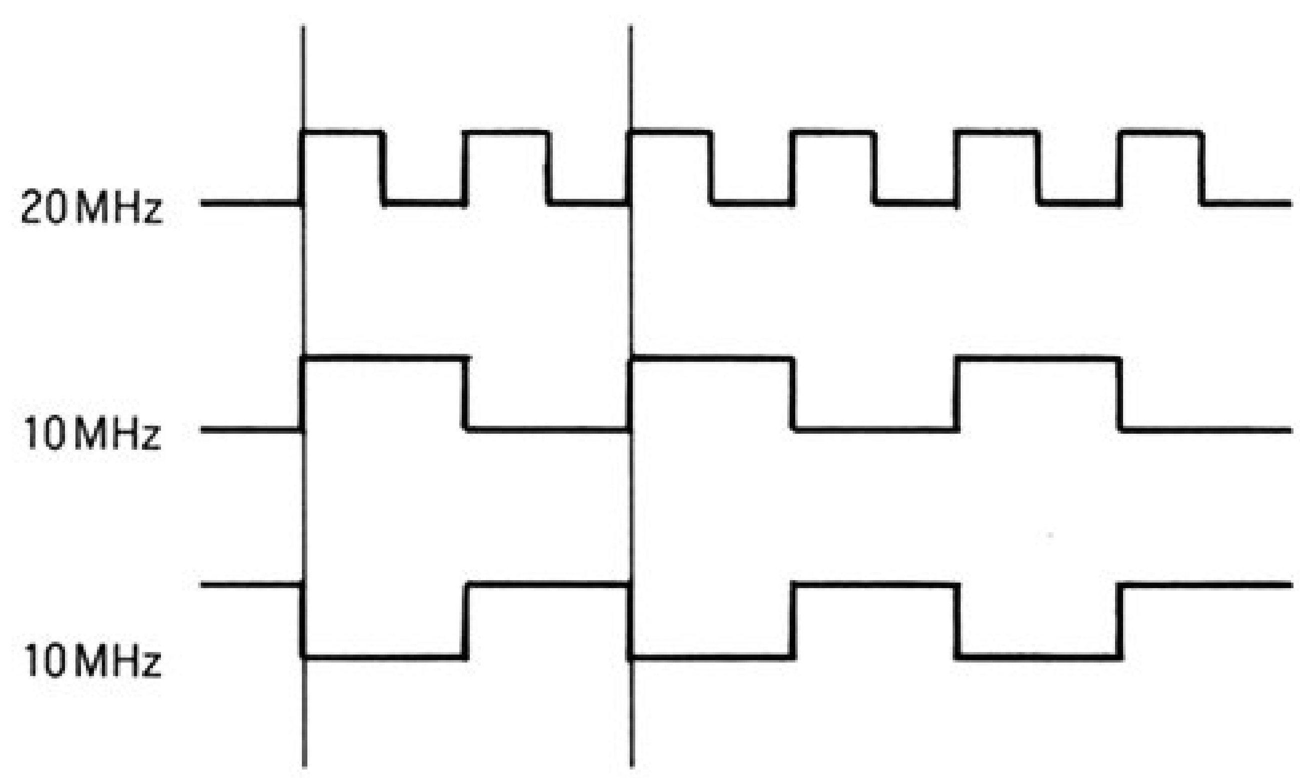
拡張スロットに出ている信号のうちアドレスバスやデータバスなど，CPUから供給される信号は本体内部のLSIに配線されている信号をそのまま引き出しているため，双方向での使用を可能にしています。

＊1：C-MOSのLSIなどと直結されている信号の場合，出力段はFETですから，正確にはオープンコレクタではなく，オープンドレインと呼ぶべきですが，ここではオープンコレクタの名称で統一しておきます。

1 クロック関係（出力：トーテムポール）

X68000の拡張スロットには10MHzのCPUクロック，およびその反転クロック，20MHzのクロックの3種類のクロック信号が出力されています（図4）。10MHzのクロックは

●図……4　クロック信号



●表……1　拡張スロットの信号

番号	A側			B側		
	信号名	I/O	内容	信号名	I/O	内容
1	GND	…	グランド	GND	O	グランド
2	20MHz	O	クロック(20MHz)	10MHz	O	クロック(10MHz)
3	GND	…	グランド	$\overline{\text{10MHz}}$	O	反転クロック(10MHz)
4	DB0	I/O	データバス	E	O	6800系周辺デバイス用クロック
5	DB1	I/O		AB1	I/O	アドレスバス
6	DB2	I/O		AB2	I/O	
7	DB3	I/O		AB3	I/O	
8	DB4	I/O		AB4	I/O	
9	DB5	I/O		AB5	I/O	
10	DB6	I/O		AB6	I/O	
11	GND	…	グランド	GND	…	グランド
12	DB7	I/O	データバス	AB7	I/O	アドレスバス
13	DB8	I/O		AB8	I/O	
14	DB9	I/O		AB9	I/O	
15	DB10	I/O		AB10	I/O	
16	DB11	I/O		AB11	I/O	
17	DB12	I/O		AB12	I/O	
18	DB13	I/O		AB13	I/O	
19	DB14	I/O		AB14	I/O	
20	DB15	I/O		AB15	I/O	
21	GND	…	グランド	GND	…	グランド
22	＋12V	O	＋12V電源	AB16	I/O	アドレスバス
23	＋12V	O		AB17	I/O	
24	FC0	I/O	ファンクションコード	AB18	I/O	
25	FC1	I/O		AB19	I/O	
26	FC2	I/O		AB20	I/O	
27	$\overline{\text{AS}}$	I/O	アドレスストローブ	AB21	I/O	
28	$\overline{\text{LDS}}$	I/O	下位データストローブ	AB22	I/O	
29	$\overline{\text{UDS}}$	I/O	上位データストローブ	AB23	I/O	
30	R/$\overline{\text{W}}$	I/O	リード/ライト('L'＝Write)	IDDIR	O	データバスバッファ方向制御
31	…		未使用	…		未使用
32	－12V	O	－12V電源	HSYNC	O	水平同期信号
33	－12V	O		VSYNC	O	垂直同期信号
34	$\overline{\text{VMA}}$	O	アドレスバス有効	$\overline{\text{DONE}}$	O	DMAブロック転送終了
35	$\overline{\text{EXVPA}}$	I	6800系周辺デバイスアクセス信号	$\overline{\text{DTC}}$	I	DMAデータ転送終了
36	$\overline{\text{DTACK}}$	I/O	データ転送終了	$\overline{\text{EXREQ}}$	I/O	DMAデータ転送要求
37	$\overline{\text{EXRESET}}$	O	外部リセット	$\overline{\text{EXACK}}$	O	DMAデータ転送許可
38	$\overline{\text{HALT}}$	I/O	CPUホールト要求/システム停止表示	$\overline{\text{EXPCL}}$	I/O	DMA汎用入出力信号
39	$\overline{\text{EXBERR}}$	I/O	バスエラー	$\overline{\text{EXOWN}}$	I/O	バス解放動作用信号
40	$\overline{\text{EXPWON}}$	I	電源ON要求	$\overline{\text{EXNMI}}$	I	NMI要求信号
41	GND	…	グランド	GND	…	グランド
42	Vcc 2	O	バックアップ電源(＋5V)	$\overline{\text{IRQ2-n}}$	O	レベル2割り込み要求
43	Vcc 2	O		$\overline{\text{IRQ4-n}}$	O	レベル4割り込み要求
44	SELEN	O	ロウ／カラムアドレス切り替え	$\overline{\text{IACK2-n}}$	O	レベル2割り込みアクノリッジ
45	CASRDEN	O	メインメモリCAS信号(リード時)	$\overline{\text{IACK4-n}}$	O	レベル4割り込みアクノリッジ
46	CASWRL	O	下位データ書き込み用CAS信号	$\overline{\text{BR-n}}$	O	バスリクエスト
47	CASWRU	O	上位データ書き込み用CAS信号	$\overline{\text{BG-n}}$	O	バスグランド
48	INH2	O	リフレッシュサイクル表示信号	$\overline{\text{BGACK}}$	O	バスグランドアクノリッジ
49	Vcc 1	O	＋5V電源	Vcc 1	O	＋5V電源
50	Vcc 1	O		Vcc 1	O	

いずれも20 MHzの立ち上がりに同期して反転します。各クロック間の位相関係は極力正しくなるよう工夫されているため，拡張ボード側ではクロックの分周や反転に伴う遅れを気にする必要はほとんどありません。

2 データバス(DB0〜DB15;入出力：トライステート)

CPUのデータバスとバッファを一段経由して接続されている16ビットバスです。CPUとのデータ入出力はこのラインを経由して行います。IBMのPC/ATなどでは，過去に存在した8ビットバスとの互換性を保つため，基本的に8ビットバスとして動作し，拡張ボード側が16ビットアクセスが可能なことを本体に通知すると16ビットバスとして動作するようになっているのですが，X68000の場合はそのような過去はありませんので，必ず16ビットバスとして動作します。

3 アドレスバス(AB1〜AB23;入出力：トライステート)

CPUのアドレスバスにバッファを一段通して接続されている24ビットのアドレスバスです。

4 ファンクションコード(FC0〜FC2;入出力：トライステート)

CPUの出力するファンクションコードが出力されています（表2）。

●表……2　ファンクションコード

FC2	FC1	FC0	意　味
0	0	0	未定義
0	0	1	ユーザ・データ
0	1	0	ユーザ・プログラム
0	1	1	未定義
1	0	0	未定義
1	0	1	スーパバイザ・データ
1	1	0	スーパバイザ・プログラム
1	1	1	割り込みアクノリッジ

5 AS，R/W，LDS，UDS（入出力：トライステート）

それぞれ，CPUのバス制御信号です。ASは有効なアドレスが載っていることを外部に示す信号，R/WはCPUがリード動作/ライト動作のいずれを行おうとしているのかを示す信号，LDS/UDSはそれぞれデータバスの下位8ビット，上位8ビットを使用しようとしていることを示す信号です。

6 DTACK（入出力：オープンコレクタ）

CPUのDTACK入力ピンに接続される信号です。オプションボードがデータ転送を終了した(リード時は取り込みが完了した，ライト時は書き込み動作が終了した)ことをCPU側に通知し，バスサイクルを完了させます。DMA転送時は，必ず1ウェイト入るようになっていますので，オプションボードでDMA転送を利用する場合は気をつけてください。X68000はバスのハングアップを監視する回路があり，AS信号が'L'になってから，9μS以内にDTACK信号を返さないとバスエラーを発生させます。9μS以上のアクセス時間がかかるような回路を作らないようにしてください。

7 EXVPA（入力：オープンコレクタ）

CPUのVPA入力ピンに接続される信号です。モトローラの8ビットCPUである6800のファミリーのLSIを接続したときに使われる信号です。アクセスされた時点でこの信号を返すと，CPUは8ビットバス相当の動作をするとともにバス信号をE信号に同期したものとします。6800 CPUはもちろん，ファミリーLSIもかなり古いデバイスであり，オプションボードで使うこともほとんどないでしょう。

8 VMA（出力：トライステート

CPUのVMA信号出力ピンに接続される信号です。VPA信号によって6800ファミリーデバイスが接続されていることが示されると，アドレスバス上に有効なアドレスが載っており，プロセッサがE信号に同期して動作していることを示すためにこの信号が使用されます。

9 E（出力：トーテムポール）

CPUのE信号出力ピンに接続される信号です。6800ファミリーデバイスのためのイネーブ

ル信号です。E信号の周期は10クロックサイクル（6クロック間'LOW', 4クロック間'HIGH'）になっています。

10 EXRESET（出力：トーテムポール）

システムのリセット信号です。この信号がアクティブ('Low'レベル)になる要因と，その期間は次のようになっています。

・パワーON時
　Vcc1がONした後，約200 mS～300 mSの間アクティブになります。

・RESETスイッチを押したとき
　押された後約20 μSの間アクティブになります。

・CPUがリセット命令を実行したとき
　実行後，約12.4 μSの間アクティブになります。

11 HALT（入出力：オープンコレクタ）

CPUのHALTピンと接続される信号です。外部からアクティブにすると，CPUは現在実行しているバスサイクルが終了した時点で動作を停止します。CPUが出力する信号はすべてインアクティブ状態になり，トライステート出力の信号はすべてハイ・インピーダンス状態になります。

二重バスエラーの発生などにより，CPUが自分で動作を停止した場合には，この信号がアクティブとなり，外部デバイスに対してCPUが停止したことを示します。

12 EXBERR（入出力：オープンコレクタ）

本体内部で発生するバスエラー信号が引き出されています。本体からの出力信号とみなせば本体内部でバスエラーが発生したことを示す信号として，本体への入力信号としてみれば，オプションボードからCPUにバスエラーの発生を伝えるための信号として利用できます。

13 EXPWON（入力：オープンコレクタ）

X68000の電源をONさせる信号です。本体正面の電源スイッチがOFF状態のとき(Vcc2のみ生きている状態のとき）にこの信号をアクティブにすると本体の電源が入ります。この状態でこの信号をインアクティブにすると，CPUに対して割り込みが発生し，CPUがシステム

ポートを使って電源 OFF 処理をした時点で電源が落ちます。

EXPWON 信号は最低でも 500 mS 以上アクティブにするようにしてください。

14 IDDIR（出力：トーテムポール）

リード動作時 'Low'，ライト動作時 'High' になる信号です。データバスバッファの方向制御信号として利用できます。

15 拡張メモリボード用制御信号（出力：トーテムポール）

拡張スロットにメモリ増設が容易にできるよう，SELEN，CASRDEN，CASWRL，CASWRU の 4 本の信号が用意されています。

SELEN はメモリアクセスサイクルであることを示す信号です。メモリアクセスサイクルのとき'High'になり，それ以外のときは'Low'になっています。

CASRDEN はリードサイクルであるとき，CASWRL，CASWRU はそれぞれ下位バイト，上位バイトへのライトサイクルであるときに'High'になります。信号の名称や，AS がアクティブになってからある程度時間をおいて動くところからみて，D-RAM の CAS 信号を作るときに使用することを考慮したものなのでしょうが，純正のメモリボードの回路図を見る限りでは，これらの信号は使用されていません。

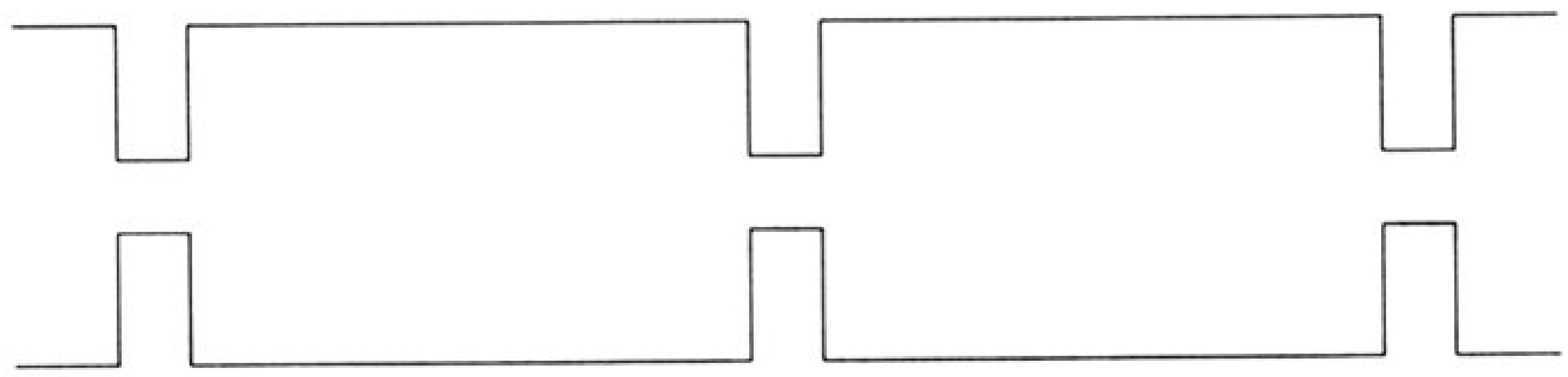
16 INH 2（出力：トーテムポール）

システムメモリのリフレッシュサイクルにあることを示す信号です。リフレッシュサイクルは約 14 μS おきに発生しています。AS 信号が'Low'になったとき，この信号が'Low'であればリフレッシュサイクルです。

17 同期信号（出力：トーテムポール）

HSYNC と VSYNC はそれぞれ CRT ディスプレイ用の水平同期信号，垂直同期信号です。この信号は CZ-600 C（元祖タイプ）では負論理（同期期間中，'Low'になる），それ以降の機種では正論理（同期期間中，'High'になる）と反転していますので注意してください（図 5）。

●図……5　機種による同期信号の違い



18 DONE（入出力：オープンコレクタ）

DMAコントローラのDONE入出力端子に接続されている信号で，外部デバイスやDMAコントローラ自身が，現在転送中のデータがブロックの最終データであることを示すために使われます。外部デバイスがこの信号をアクティブにすると，現在転送中のデータをブロックの最終としてDMAコントローラが受け取りますし，DMAコントローラは自身のカウントレジスタの値が0になった(設定されたバイト数分の転送が終了した)時点でDONE信号をアクティブにします。

19 DTC（出入：トライステート）

DMAコントローラのDTC端子と接続されています。DMAコントローラが外部に対してデータ転送動作が正常完了したことを示す信号です。

20 EXREQ（入力：オープンコレクタ）

DMAコントローラのチャンネル#2の転送要求信号(REQ 2)に接続されています。X 68000のDMAコントローラは4つのチャンネルを持っていますが，残りの3つのチャンネルはADPCM，フロッピーディスク，ハードディスク用として使われていますので，拡張スロットで使用できるのはチャンネル#2だけです。複数のスロットから同時にDMA要求を出されても，DMAコントローラはどちらからの要求であるか判断できませんので，同時に使用することはできません。

21 EXACK（出力：トーテムポール）

DMAコントローラのチャンネル#2の転送アクノリッジ信号（ACK 2）に接続されています。チャンネル#2の転送中であることを外部に示すために使われる信号です。DMAコントロ

ーラをシングルアドレスモードで使うときは，オプションボード側はこの信号がアクティブになったのを見てデータ入出力を行うことになります。

X 68000 では基本的に DMA コントローラをデュアルアドレスモードで使うことになっていますので，この信号を使う必要性はほとんどないでしょう。

22 EXPCL（入出力：トライステート）

DMA コントローラのチャンネル#2 のペリフェラルコントロール信号（PCL 2）に接続されています。この信号はステータス入力信号，転送スタートパルス出力信号，アボート入力など，さまざまな目的に使用することができます。詳細は拙著『Inside X 68000』などを参照ください。

23 EXOWN（入出力：オープンコレクタ）

CPU 以外のデバイスがバスを使用するとき，アクティブにする信号です。DMA コントローラがバスを使用しているとき（転送動作を行っているとき）アクティブになります。オプションボード側でバスを使用するときも，この信号をアクティブにするようにします。

24 EXNMI（入力：オープンコレクタ）

CPU の NMI(レベル 7 割り込み)要求信号になります。レベル 7 割り込みは最もレベルの高い割り込み要求で，CPU のステータスレジスタの割り込みマスクビットによってマスクすることもできません。

X 68000 では NMI はハード的にオートベクタ方式で使うようになっていますので，EXNMI に対する割り込みアクノリッジサイクルで DTACK や EXVPA 信号などをアクティブ（'Low'レベル）にしてはなりません。

25 割り込み要求信号（IRQn−m；入力：オープンコレクタ）

IRQ 2-1/IRQ 2-2，IRQ 4-1/IRQ 4-2 はそれぞれ割り込みレベル 2，および 4 への割り込み要求信号です。-1，-2 となっているのは，スロットごとに独立した配線が行われているため区別をしているもので，各スロット単位で見れば単にレベル 2 とレベル 4 の割り込みラインが 1 本ずつあるだけです。

26 割り込み応答信号（IACKn−m；出力：トーテムポール）

IACK 2-1/IACK 2-2, IACK 4-1/IACK 4-2 はそれぞれ割り込みレベル 2，および 4 の割り込み要求に対する応答信号です。-1, -2 となっているのは要求信号と同様，スロットごとに独立した配線が行われているため区別されているだけのことです。

27 バスリクエスト信号（BR−1，BR−2；入力：オープンコレクタ）

CPU に対するバス解放要求信号です。スロットごとに独立した配線が行われているので，それぞれ-1, -2 を付けて区別しています。

28 バスグラント信号（BG−1，BG−2；出力：トーテムポール）

CPU からのバス解放表示信号です。解放要求信号と同様，スロットごとに独立した配線が行われているので，それぞれ-1, -2 を付けて区別しています。

29 BGACK（入出力：トライステート）

CPU の BGACK 端子と接続されています。CPU 以外のデバイス（本体内部では DMA コントローラ）がバスを使用しているとき，'Low'となります。オプションボードがバスリクエスト/バスグラント信号を使ってバスを占有したときも'Low'になるように設計してください。

●4 拡張スロットの割り込み

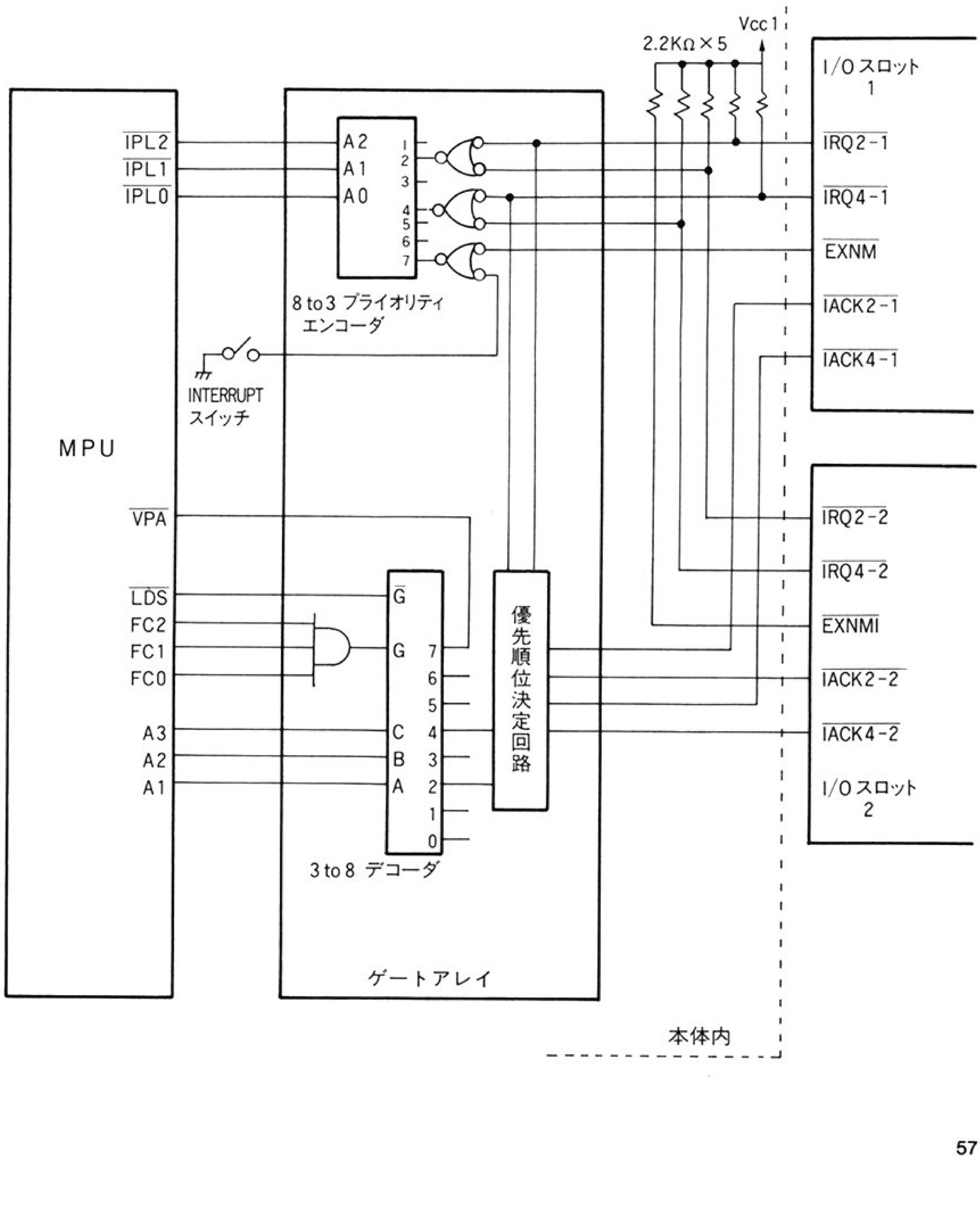
X 68000 の CPU である 68000 は，割り込みをレベル 1 からレベル 7 までの 7 段階のレベルをつけて管理できるようになっています。X 68000 では拡張スロットにレベル 2 とレベル 4 の 2 つの割り込み要求信号が設けられています。さらに，各スロットの割り込み要求信号は別々のものとして扱われているため，両方のスロットで同じレベルの割り込みを使用してもかまいません。当然，IACK（割り込み応答）信号もスロットごとに別々となっており，割り込み要求を受け付けた側のスロットの IACK 信号がアクティブになります。スロット間の優先順位はスロット 1 の方がスロット 2 よりも高く設定されており，両方のスロットから同時に要求があっ

ロット１の方がスロット２よりも高く設定されており，両方のスロットから同時に要求があった場合にはスロット１からの要求が先に処理されます。

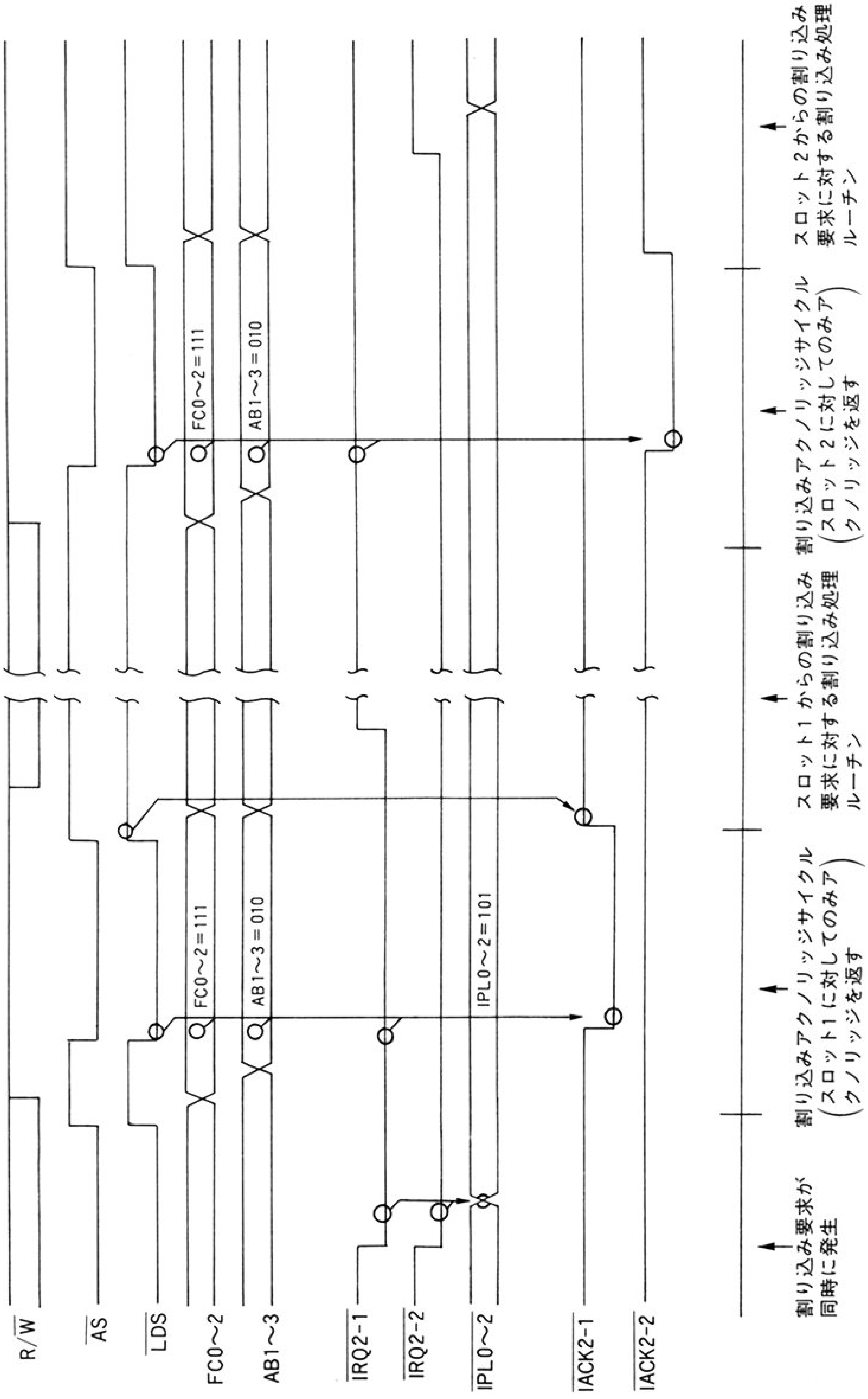
もちろん，この場合でもスロット２の割り込み要求は捨てられるわけではありません。CPUが割り込み処理を終え，次の割り込みが受け付けられる状態になるまで要求信号をアクティブにし続ければ，その時点でIACK信号が返されます。

図６に拡張スロットからの割り込み系統ブロックを，図７に割り込みシーケンスのタイミングを示しますので参考にしてください。

●図……６　I/Oスロットからの割り込みに関するブロック図



●図……7　割り込みシーケンスタイミングチャート

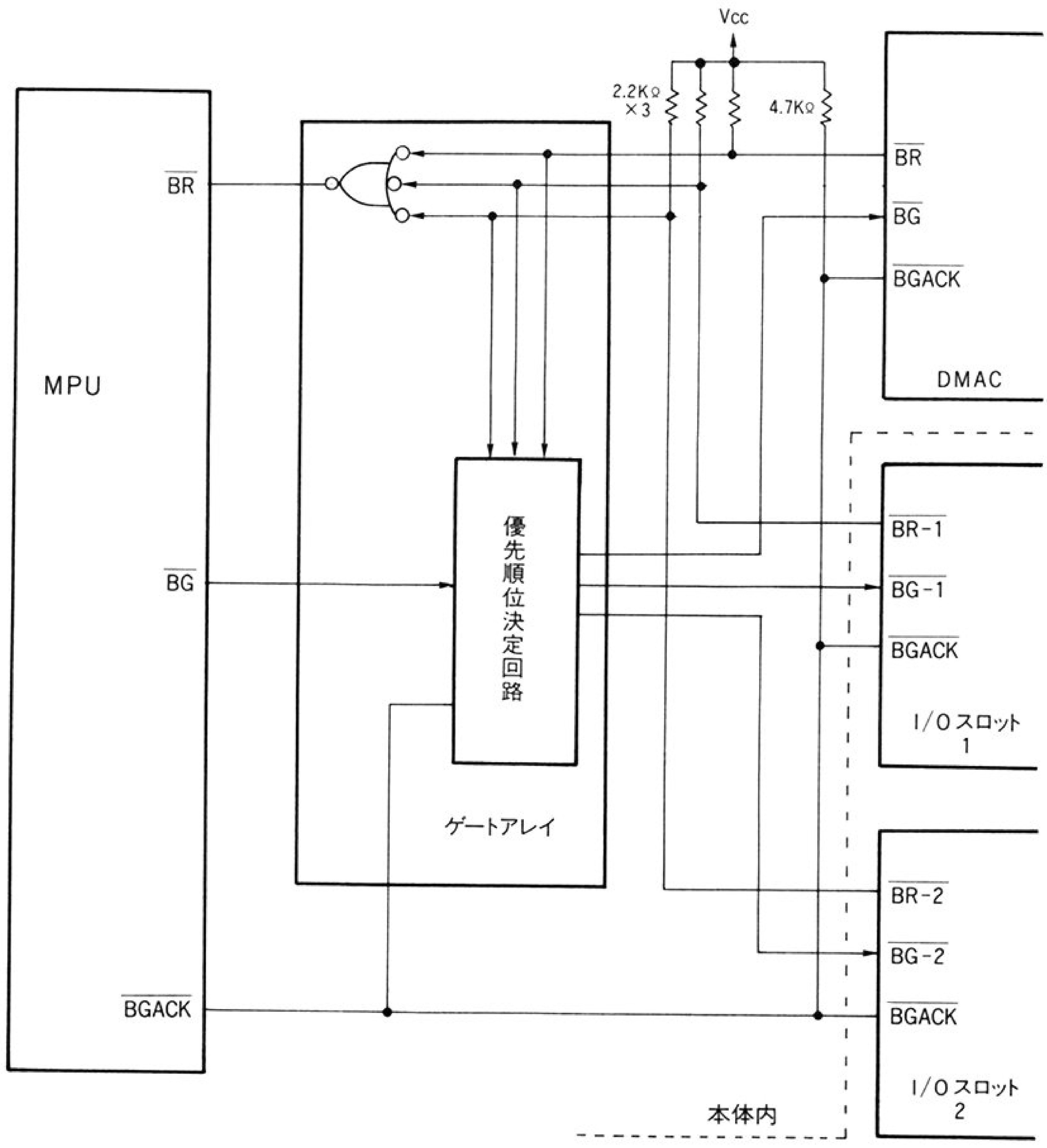


●5 拡張スロットからのバス占有

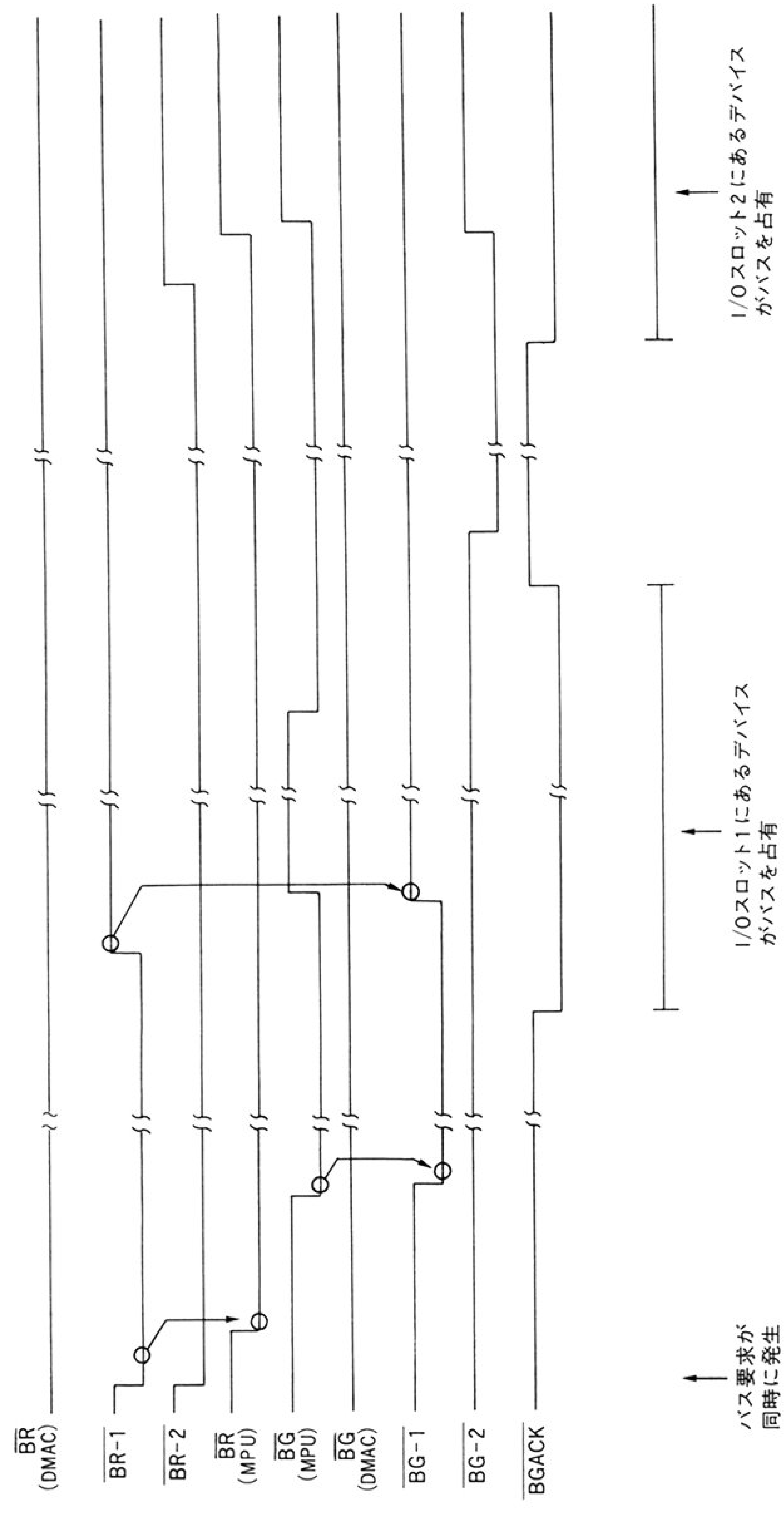
X 68000ではDMAコントローラのようにCPUにバスを解放させ，拡張ボードがバスを占有する（「乗っ取る」という言い方をする場合もあります）ことができるようになっています。

2つの拡張スロット，および本体内部のDMAコントローラからのバス解放要求信号はすべて

●図……8 I/O スロットからのバス要求に関するブロック図



●図……9　バス調停タイミングチャート



別々のものとして扱われており，優先順位はDMAが最も高く，以下拡張スロット1，拡張スロット2の順となっています。割り込みのときと同様，複数の要求が重なった場合にはこの優先順位に従って処理されますが，ほかの要求も捨てられてしまうわけではなく，受け付けられる状態になれば処理が行われますので，それまで要求を保持し続けるようにすればよいでしょう。

拡張スロットからバスを占有するには，

(1) BR信号を'Low'にしてバス解放要求を行い，
(2) BG信号が'Low'になってバス解放要求が受け付けられたことが示されたらBGACKを'Low'にしてバスを占有し，
(3) 使用が終わったらBGACK信号を'High'にしてバスの占有が終了したことをCPUに知らせる

という方法をとります。

図8にバス解放要求関係の信号系統ブロックを，図9にバス占有シーケンスのタイミングを示しますので参考にしてください。

6 I/OスロットのDC規定

X68000のI/OスロットのDC規定を表3に示します。表中，IIL/IIHはそれぞれ信号レベルが'L'のときにボードから流れ出す電流と'H'レベルのときにオプションボード内に流れ込むことができる電流の最大値を，IOL/IOHはその信号をオプションボード側でドライブするときにボード側が供給しなくてはならない電流の最小値を示しています。

オプションボードで使用するドライバ/レシーバICやプルアップ／プルダウン抵抗は，これらの電流規格を満足するようなものを選択するようにしてください。

また，I/Oスロットから取り出した信号は極力短い距離でバッファICで受けてから，ボード内を引き回すようにしてください。I/Oスロットからの配線長が長いと他の信号とのクロストークや信号の反射，配線ディレイなどによりボード単体だけでなくシステム全体の誤動作を招きやすくなります。

●表……3　I/O スロットのＤＣ規定

<table>
<tr><th>信号名</th><th>I_{IL}(mA)</th><th>I_{IH}(μA)</th><th>I_{OL}(mA)</th><th>I_{OH}(mA)</th></tr>
<tr><td>AB1～AB23</td><td rowspan="8">−2.0</td><td rowspan="8">50</td><td rowspan="8">24</td><td rowspan="8">−3.0</td></tr>
<tr><td>DB0～DB15</td></tr>
<tr><td>AS</td></tr>
<tr><td>LDS</td></tr>
<tr><td>UDS</td></tr>
<tr><td>R/W</td></tr>
<tr><td>FC0～FC2</td></tr>
<tr><td>VMA</td></tr>
<tr><td>DTACK</td><td>−0.4</td><td>300</td><td rowspan="2">8</td><td rowspan="20">——</td></tr>
<tr><td>EXVPA</td><td>——</td><td>合計で
600</td></tr>
<tr><td>E</td><td>−1.2</td><td>60</td><td rowspan="2">——</td></tr>
<tr><td>10MHz
$\overline{\text{10MHz}}$
20MHz</td><td>−2.0</td><td>500</td></tr>
<tr><td>HALT</td><td>合計で
−0.2</td><td>合計で
250</td><td>8</td></tr>
<tr><td>EXRESET</td><td>−1.2</td><td>60</td><td>——</td></tr>
<tr><td>EXBERR</td><td>−0.4</td><td>300</td><td rowspan="2">8</td></tr>
<tr><td>EXOWN</td><td>——</td><td>200</td></tr>
<tr><td>IDDIR</td><td>−0.8</td><td>40</td><td>——</td></tr>
<tr><td>HSYNC</td><td rowspan="2">−0.4</td><td rowspan="2">60</td><td>——</td></tr>
<tr><td>VSYNC</td><td>——</td></tr>
<tr><td>SELEN</td><td rowspan="5">−3.0</td><td rowspan="5">300</td><td rowspan="5">——</td></tr>
<tr><td>CASRDEN</td></tr>
<tr><td>CASWRL</td></tr>
<tr><td>CASWRU</td></tr>
<tr><td>INH2</td></tr>
<tr><td>DONE</td><td>−1.2</td><td>120</td><td>8</td></tr>
<tr><td>DTC</td><td>−0.8</td><td>60</td><td>——</td></tr>
<tr><td>EXREQ</td><td>——</td><td>120</td><td>8</td></tr>
<tr><td>EXACK</td><td rowspan="3">−0.8</td><td rowspan="2">60</td><td>——</td></tr>
<tr><td>EXPCL</td><td rowspan="4">8</td><td>1.0</td></tr>
<tr><td>EXOWN</td><td rowspan="6">120</td><td rowspan="6">——</td></tr>
<tr><td>IRQ2-1～IRQ4-2</td><td rowspan="2">——</td></tr>
<tr><td>EXNMI</td></tr>
<tr><td>IACK2-1～IACK4-2</td><td>−0.8</td><td>——</td></tr>
<tr><td>BR-1, BR-2</td><td>——</td><td>8</td></tr>
<tr><td>BG-1, BG-2</td><td rowspan="2">−0.8</td><td>——</td></tr>
<tr><td>BGACK</td><td>60</td><td>8</td><td>−1.0</td></tr>
</table>

電源容量（1スロットあたり）

Vcc 1（+5V）	600 mA
Vcc 2（+5V）	30 mA
Vcc 3（+12V）	30 mA
Vcc 4（−12V）	30 mA

●7 I/OスロットのACタイミング

X 68000の拡張スロットの動作は基本的にリードサイクルとライトサイクル，メモリリフレッシュサイクルの3つに分類できます。DMAによる転送もX 68000の場合にはデュアルアドレスモードを使用しますので，CPUによる動作とほとんど変わるところはありません。CPUによるアクセスのときはEXOWN信号が'H'，DMAコントローラによるサイクルではEXOWN信号が'L'になりますので，どちらによるアクセスサイクルか区別することができます。

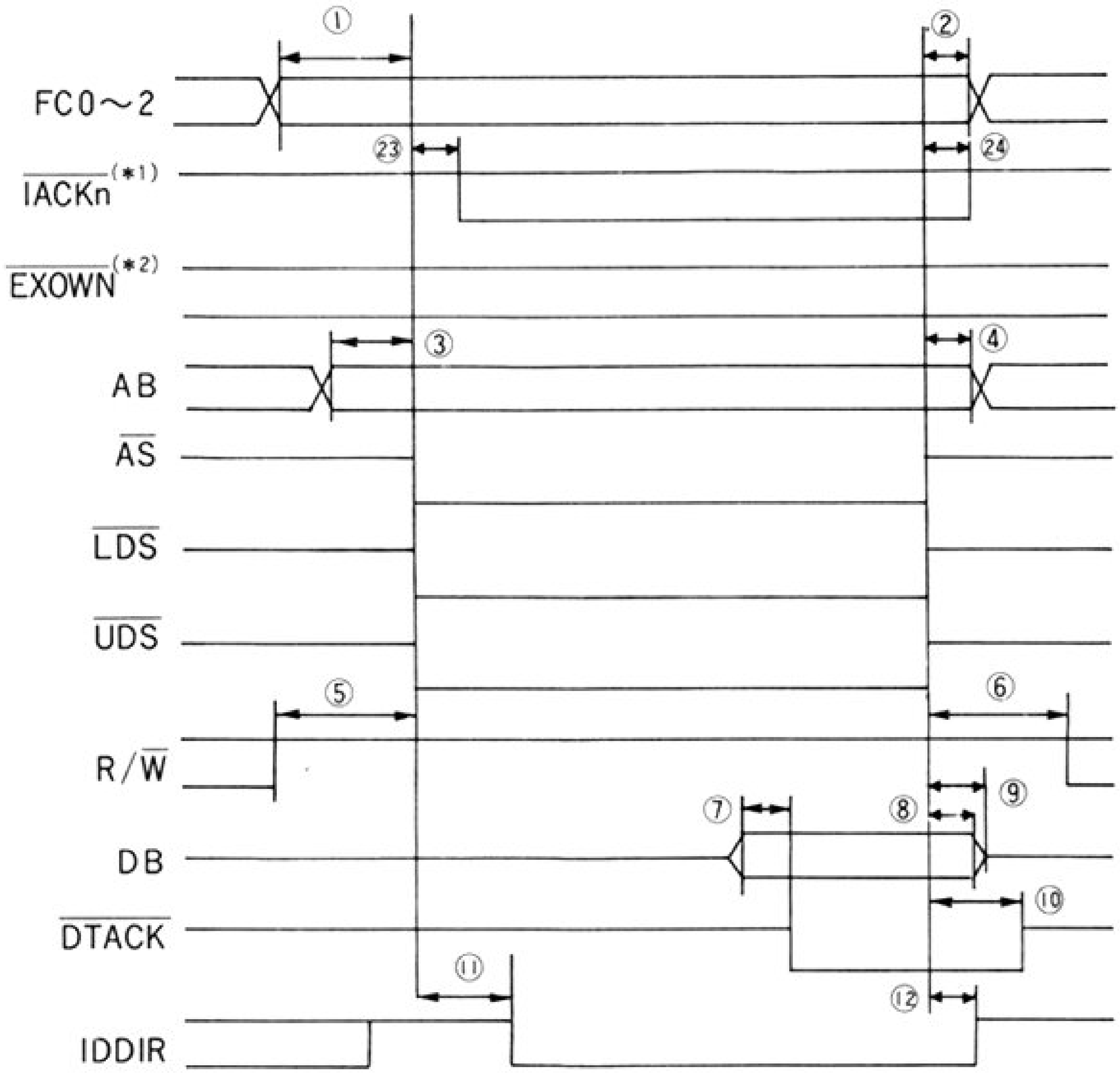
リード/ライトのタイミングを図10に示します。リード時の動作は次のようになります。

(1) CPUやDMAコントローラ(以下，単にCPUとします)のアクセスする空間やアドレスを示すFCnやABnを確定させた後，AS(アドレスストローブ)，UDS/LDS(上位データストローブ/下位データストローブ)信号を'L'にします。このときR/W信号はリード状態を示すために'H'になっており，またIDDIRは'L'になります。
(2) アクセスされた側(オプションボード側)はバス上にデータを載せた後，DTACK信号を'L'にしてCPUに有効なデータがバス上に用意できていることを示します。
(3) CPUはDTACKを受け取るとAS, UDS, LDSを'H'に戻しますので，これを見てオプションボード側はDTACK信号を'H'に戻します。

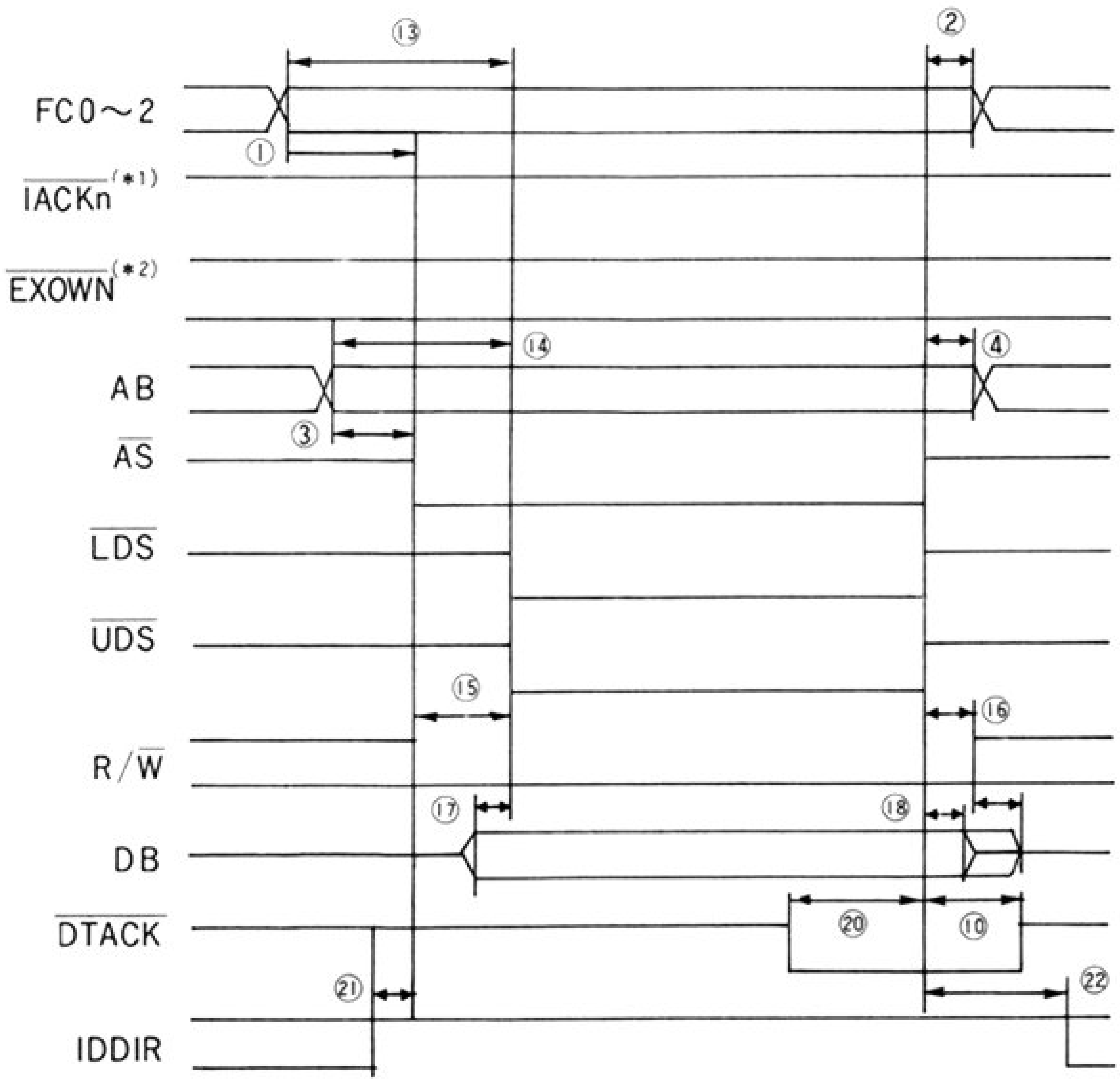
割り込みアクノリッジサイクルで，CPUにベクタを読ませるような場合もバスサイクルはIACKn信号が'L'になるほかはCPUによるリードサイクルとほとんど同じ動作となります。

ライトサイクル時は，ASによってアドレス，アクセス空間が示された後，CPUからの書き込みデータがバス上に載せられ，UDS, LDSが'L'になります。このとき，R/W信号は'L'に，IDDIRは'H'になります。

●図……10　リード/ライトサイクルタイミングチャート



(a) リードサイクル



(b) ライトサイクル

＊1　割り込みアクノリッジサイクルでは IACKn 信号が 'L' になります。(nは割り込みレベルおよびスロット番号)

＊2　MPUサイクルでは EXOWN 信号は 'H' に，DMACサイクルでは 'L' になります。

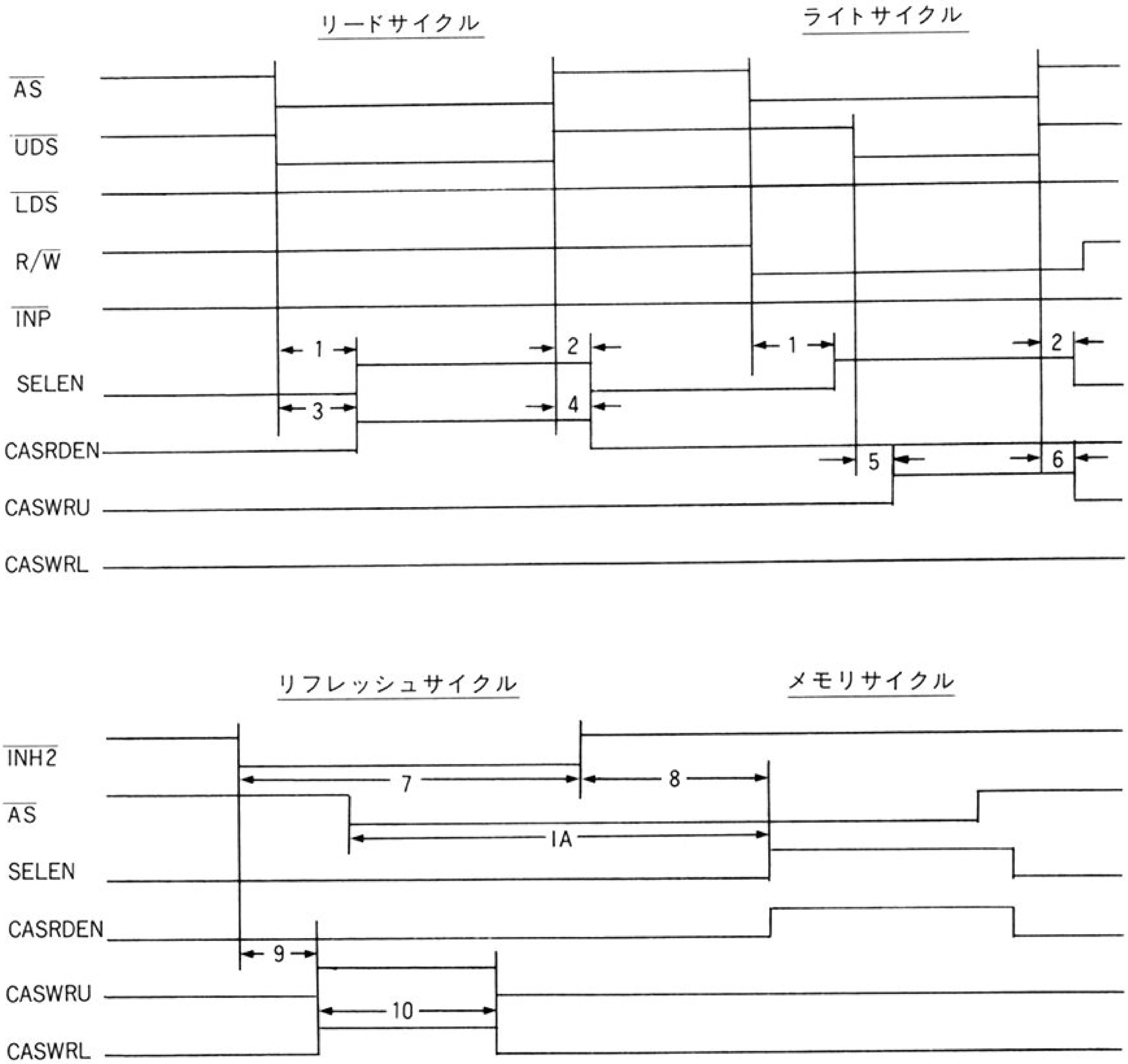
●表…… 4　I/O スロットのＡＣ特性

番号	項　　目	遅延時間(単位:ns) 最小	最大	備 考
1	FC確定からAS, UDS, LDS＝'L' まで	10		
2	AS, UDS, LDS＝'H' からFC有効期間	10		
3	アドレス確定からAS, UDS, LDS＝'L' まで	10		
4	AS, UDS, LDS＝'H' からアドレス有効期間	10		
5	R/W＝'H' からAS, UDS, LDS＝'L' まで	30		
6	AS, UDS, LDS＝'H' からR/W＝'L' まで	80		
7	データ確定からDTACK＝'L' まで	0		
8	DS＝'H' からデータホールド期間	0		
9	DS＝'H' からデータ＝'Z' まで		80	
10	DS＝'H' からDTACK＝'Z' まで	130		
11	AS, UDS, LDS＝'L' からIDDIR＝'L' まで		150	
12	AS, UDS, LDS＝'H' からIDDIR＝'H' まで	45	150	
13	FC確定からDS＝'L' まで	90		
14	アドレス確定からDS＝'L' まで	45		
15	R/W＝'L' からDS＝'L' まで	40		
16	AS, UDS, LDS＝'H' からR/W＝'H' まで	10		
17	データ確定からDS＝'L' まで	−10		
18	DS＝'H' からデータ有効期間	10		
19	R/W＝'H' からデータ＝'Z' まで		35	
20	DTACK＝'L' からAS, UDS, LDS＝'H' まで	80	300	
21	IDDIR＝'H' からAS, R/W＝'L' まで	10		
22	AS, UDS, LDS＝'H' からIDDIR＝'L' まで	180		
23	UDS, LDS＝'L' からIACKn＝'L' まで		40	
24	UDS, LDS＝'H' からIACKn＝'H' まで	5	40	

8 リフレッシュ/D-RAM用信号タイミング

リフレッシュやD-RAM制御に便利なように設けられている信号群のタイミングを図11に示します。

●図……11　メモリリード/ライトサイクルタイミングチャート



番号	項　　　目	遅延時間(単位：ns)		備考
		最小	最大	
1	AS/UDS/LDS＝'L'からSELEN＝'H'まで	40	120	
1A	〃　　（リフレッシュサイクル時）	100N＋40	100N＋120	N:ウェイト数
2	AS/UDS/LDS＝'H'からSELEN＝'L'まで	0	40	
3	AS/UDS/LDS＝'L'からCASRDEN＝'H'まで	40	120	
4	AS/UDS/LDS＝'H'からCASRDEN＝'L'まで	0	40	
5	UDS/LDS＝'L'からCASWRU/CASWRL＝'H'まで	0	80	
6	AS/UDS/LDS＝'H'からCASWRU/CASWRL＝'L'まで	0	30	
7	INH2パルス幅	390	410	
8	INH2＝'H'からCASRDEN/SELEN＝'H'まで	100M＋90	100M＋120	M＝0or1
9	INH2＝'L'からCASWRU/CASWRL＝'H'まで	75	110	
10	CASWRU/CASWRLパルス幅（リフレッシュサイクル）	190	210	

周辺機器編

オプションボード

X68000は本体に標準装備された機能が非常に多い機械です。当初はオプションボードの必要性を感じないほどでしたが，今では純正のオプションでもさまざまなものが揃ってきました。ここではシャープの純正ボードについてまとめました。

1 はじめに

オプションボードは拡張スロットに挿入され，本体の持っていない機能や本体だけでは不十分な機能を補うものです。ハードに興味を持たれた方ならば，一度はオプションボードの自作に挑戦してみたいと思ったことがあるでしょう。拡張スロットを使用すると，RS-232 Cやジョイスティックポートなどを使ったときよりも大量のデータ転送を高速に処理することができるようになりますが，その分設計も難しくなります。もしも，すでに完全に動いている回路があれば，それを参考にして独自の部分だけを変更してもよいわけですし，信号の意味の取り違えや接続忘れ（DTACKを返し忘れたなど）といった初歩的なミスも防ぐことができます。

この章では，シャープから提供されているオプションボードそれぞれについて，その機能の概略と全回路図，I/Oポートのマップやレジスタなどについてまとめてみました。X68000は多くの機能を標準で内蔵しているためオプションボードの種類はさほど多くありませんが，回路の方は同じことを行うにも実に多彩な方法を使っており，自作品のための参考回路としても十分に役に立つことと思います。

本章で扱ったオプションボードの種類と型番は次のとおりです。

・内部拡張メモリボード	CZ-6BE1	元祖タイプ用
	CZ-6BE1A	ACE 用
	CZ-6BE1A(A)	ACE/PRO 用
	CZ-6BE2A	XVI 用
	CZ-6BE2D	Compact 用
	CZ-6BE2B	CZ-6BE2A/D 用追加増設メモリボード
・外部拡張メモリボード	CZ-6BE2/CZ-6BE4	2 MB/4 MB
	CZ-6BE2C/CZ-6BE4C	2 MB/4 MB
・ビデオボード	CZ-6BV1	
・GP-IB ボード	CZ-6BG1	
・ユニバーサル I/O ボード	CZ-6BU1	
・RS-232 C ボード	CZ-6BF1	
・MIDI ボード	CZ-6BM1	
・パラレルボード	CZ-6BN1	
・FAX ボード	CZ-6BC1	
・SCSI ボード	CZ-6BS1	
・数値演算プロセッサボード	CZ-6BP1/CZ-6BP1A	

2 オプションボードのI/Oアドレス

各オプションボードが使用するアドレス空間を示しておきます。アドレスが複数書いてあるものは，ボードのスイッチ設定などでいずれかを選択できるようになっていることを，アドレスの一部が X になっているものはその部分が 0～F までの範囲の値をとれるようになっていることを示しています。

各ボードが使用するアドレスはボードの種別ごとに区分けされており，ボードの設定がどのようになっていても互いに重複することがないように考慮されています。また，オプションボードのアドレスは，メモリマップ上システム I/O 空間となっている領域にとられています。サードパーティ製のボードの場合，シャープ純正ボードとコンパチブルなもの以外はユーザ I/O

空間 ($EC0000~$ECFFFF) を使用しているはずですので，これらが純正オプションボードとアドレスの衝突を起こすおそれはありません。

●表……1　オプションボードのI/Oアドレス

ボード種別	ボード名称	アドレス
数値演算プロセッサボード	CZ-6BP1/CZ-6BP1A	$E9E000～$E9E01F　$E9E080～$E9E09F
SCSIボード	CZ-6BS1	$EA0000～$EA1FFF
FAXボード	CZ-6BC1	$EAF900～$EAF95F
MIDIボード	CZ-6BM1	$EAFA00～$EAFA0F　$EAFA10～$EAFA1F
パラレルボード	CZ-6BN1	$EAFB00～$EAFB0F　$EAFB10～$EAFB1F
RS-232Cボード	CZ-6BF1	$EAFC00～$EAFC09　$EAFC10～$EAFC19
		$EAFC20～$EAFC29　$EAFC30～$EAFC39
ユニバーサルI/Oボード	CZ-6BU1	$EAFDX0～$EAFDX3　$EAFDX4～$EAFDX7
		$EAFDX8～$EAFDXB　$EAFDXC～$EAFDXF
GP-IBボード	CZ-6BG1	$EAFE00～$EAFE1F

拡張メモリ

メモリ容量の増加が確実にフリーエリア拡大につながるX68000のユーザにとってメモリ拡張は最初に行いたい機能拡張の一つでしょう。ここでは機種ごとのメモリ拡張方法と各拡張メモリボードについて説明します。

1 機種ごとの拡張方法の違い

X68000 シリーズはメインメモリを最大 12 Mバイトまで拡張することができるようになっています。このための拡張方法は機種によって違いがあります。X68000 XVI 以前の機種は本体内部での拡張は 2 Mバイトまでで，それ以上は拡張スロットにメモリボードを挿入することで行いますが，XVI 以降では本体内部で 8 Mバイトまで(標準装備の 2 Mバイト＋拡張ボードの 6 Mバイト) 拡張できるようになっています。

また，Compact では内部拡張用メモリボード上に数値演算コプロセッサ (CZ-6BP2) を取り付けるためのソケットが用意されています。

1·1 内部拡張メモリ

内部拡張メモリボードの一覧を表 1 に示します。X68000 の元祖タイプ，および ACE, PRO では本体の標準メモリは 1 Mバイトであり，拡張メモリボードでさらに 1 Mバイトを追加する

ようになっています。このためのメモリボードは元祖タイプではCZ-6BE1，ACEではCZ-6BE1A，PROではCZ-6BE1A(A)となります。CZ-6BE1A(A)は金具を工夫してACEとPROの両方に対応できるようにしたものですので，ACEでも使用可能です。

XVI/Compact用の内部拡張メモリボード(CZ-6BE2A，CZ-6BE2D)は，ボード上の2Mバイトのほかに1枚あたり2Mバイトの増設メモリモジュール（CZ-6BE2B）を2枚まで装着できるソケットが用意されており，これを使うことで最大6Mバイトのメモリボードとして使用することができるようになっています。

CZ-6BE2 A/6BE2D自身が装着されているか否かや，各ソケットにメモリモジュールが挿入されているか否かのステータスは本体のメモリコントローラLSIに伝えられ，どの空間が内部拡張エリアであるかという判断は自動的に行われます。これにより，内部拡張を使用せずに拡張スロットにメモリボードを搭載することもできるようになっています。

使用されているメモリは元祖タイプ以来の集積密度の向上をそのまま反映しています。元祖タイプではD-RAMに256Kビットのものを使っており，32個も使ってようやく1Mバイトだったものが，ACE用では1Mビット×8個となり，さらにXVI/Compactでは4Mビットのものが使われるようになり，4個で2Mバイトの記憶容量を構成できるようになっています。

●表……1　内部拡張メモリボード一覧

ボード型名	対応機種	容量
	使用メモリ	
CZ-6BE2D (CZ-6BE2B)	X68000Compact (CZ-6BP2用ソケット付き)	2MB
	HM514402　(1M×4ビット　:　70nS) × 4個	
CZ-6BE2A (CZ-6BE2B)	X68000XVI/XVI-HD	2MB
	HM514402　(1M×4ビット　:　70nS) × 4個	
CZ-6BE1A(A)	X68000ACE/ACE-HD/PRO/PRO-HD	1MB
	M5M44256AL (256K×4ビット：100nS) × 8個	
CZ-6BE1A	X68000ACE/ACE-HD	1MB
	M5M44256AL (256K×4ビット：100nS) × 8個	
CZ-6BE1	X68000	1MB
	MB81256　(256K×1ビット：120nS) ×32個	

❶·2 外部拡張メモリ

外部拡張メモリボードの一覧を表2に示します。使用しているメモリがアクセスタイム 120 nS のものから 80 nS のものに変更されており，これに伴って CAS が出るタイミングなど，若干の変更はありますが，仕様上は同じものとなっています。

CZ-6BE2C/6BE4C，CZ-6BE2/6BE4 はそれぞれ基板自体は同じものですが，搭載されているメモリ容量が 2Mバイト／4Mバイトと異なっています。この切り替えはボード上の J1 によって設定でき，ショートされていると 2Mバイト，オープンになっていると 4Mバイトが実装されているものとして扱われるようになります。

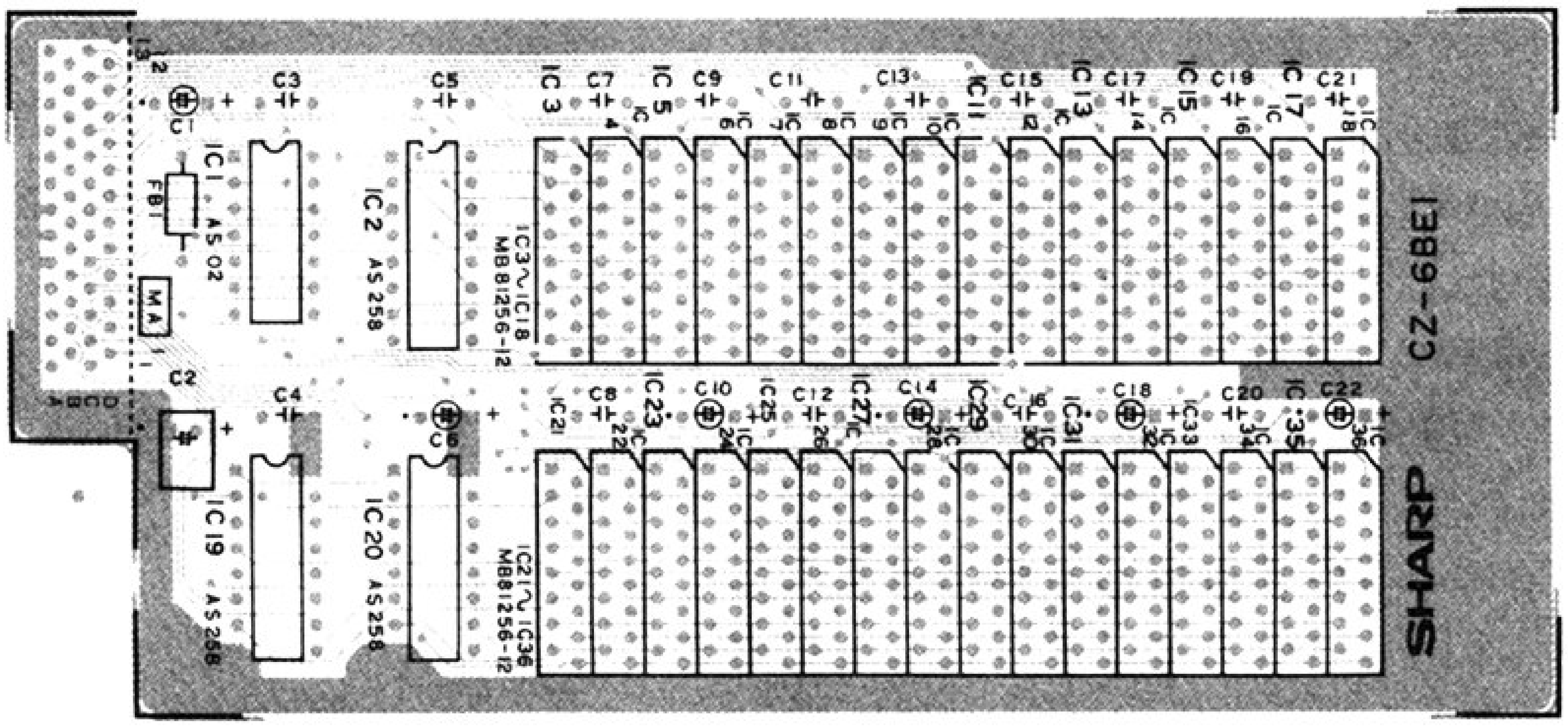
●表……2　外部拡張メモリボード一覧

ボード型名	対応機種	容量
	使用メモリ	
CZ-6BE2C/6BE4C	X68000シリーズ全機種	2MB/4MB
	MB81C1000（1M×1ビット： 80nS）×16個（2MB）/32個（4MB）	
CZ-6BE2/6BE4	X68000	2MB/4MB
	MB81C1000（1M×1ビット：120nS）×16個（2MB）/32個（4MB）	

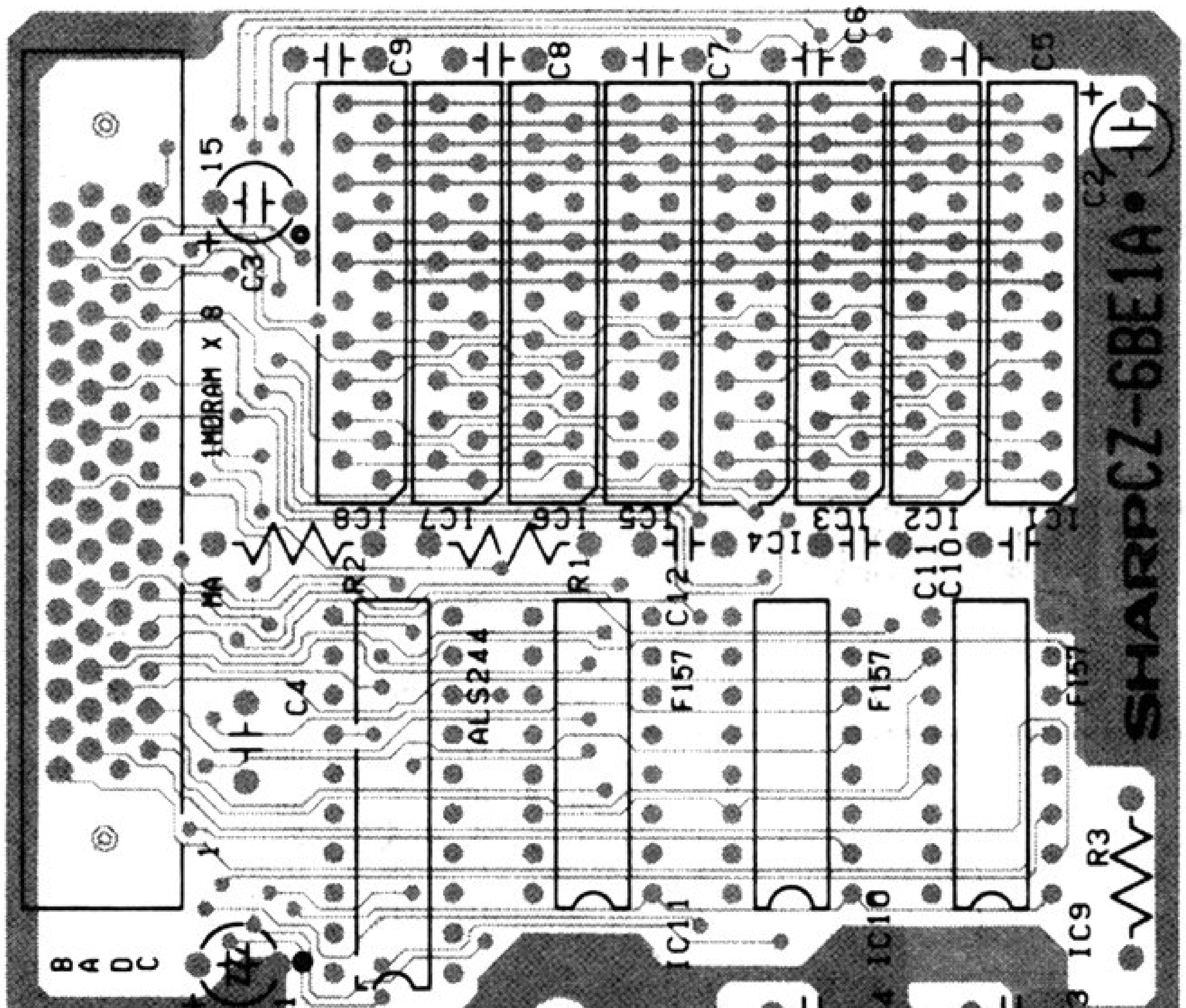
●2 部品配置

CZ-6BE1/6BE1 A（6BE1 A（A））/6BE2 A/6BE2 B/6BE2 D の部品配置を図1～図5に，CZ-6BE2（6BE4）/6BE2 C（6BE4 C）の部品配置を図6，図7に示します。

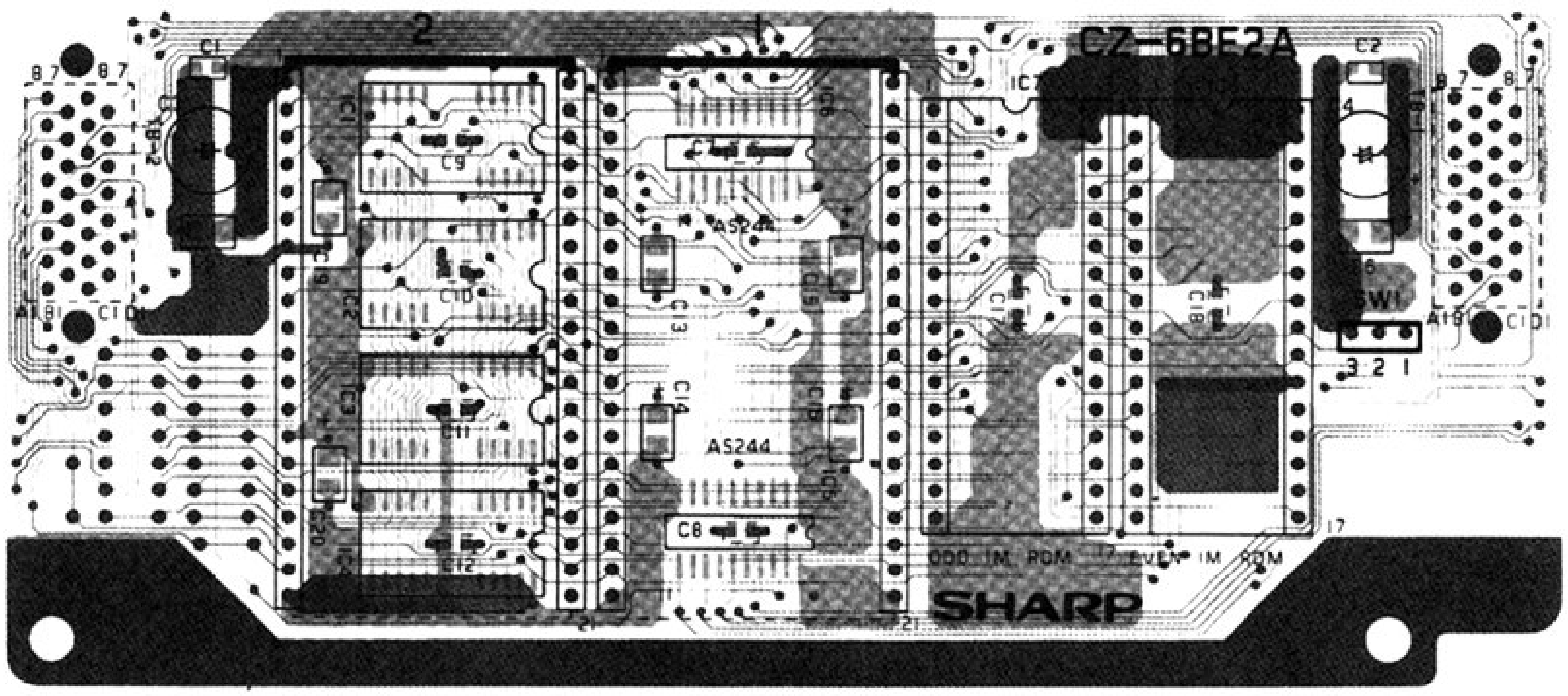
●図……1　CZ-6BE1の部品配置



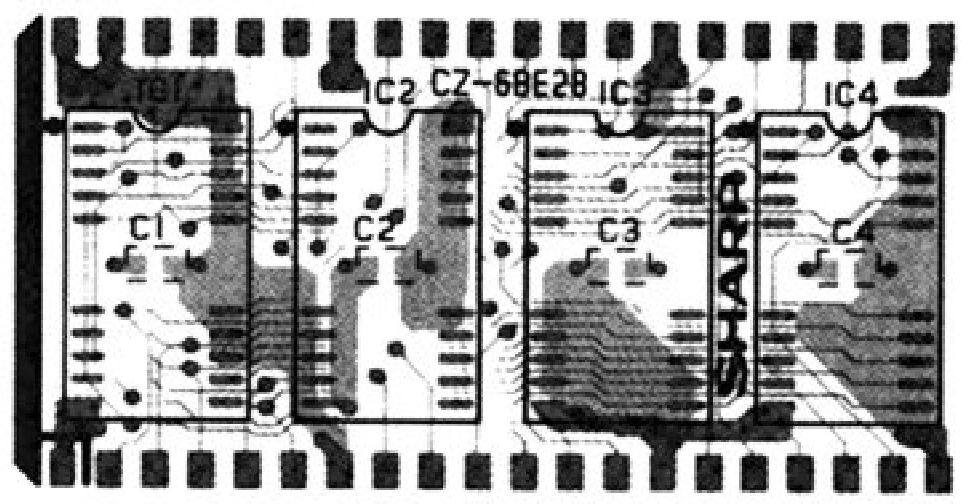
●図……2　6BE1A（6BE1A(A)）の部品配置



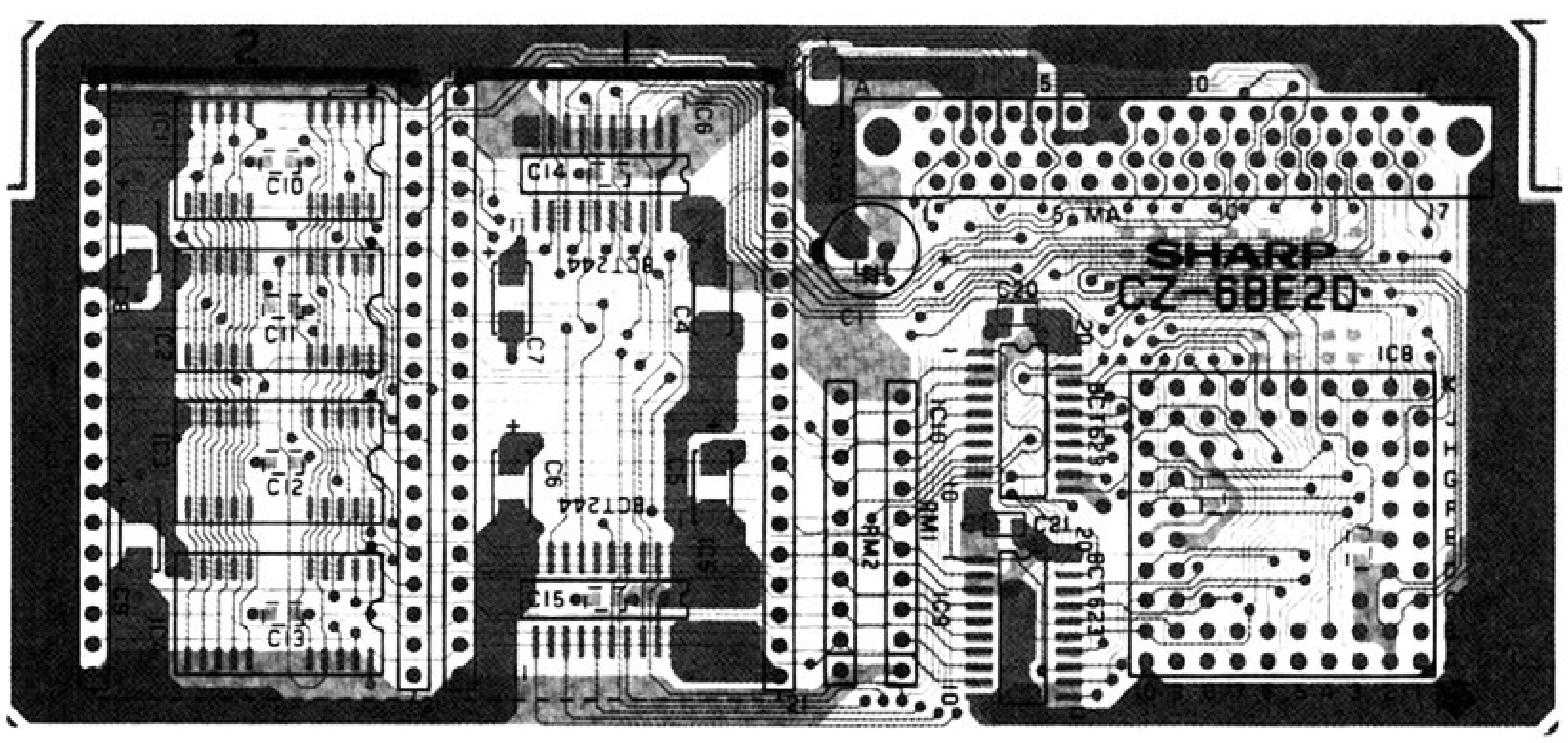
●図……3　6BE2A の部品配置



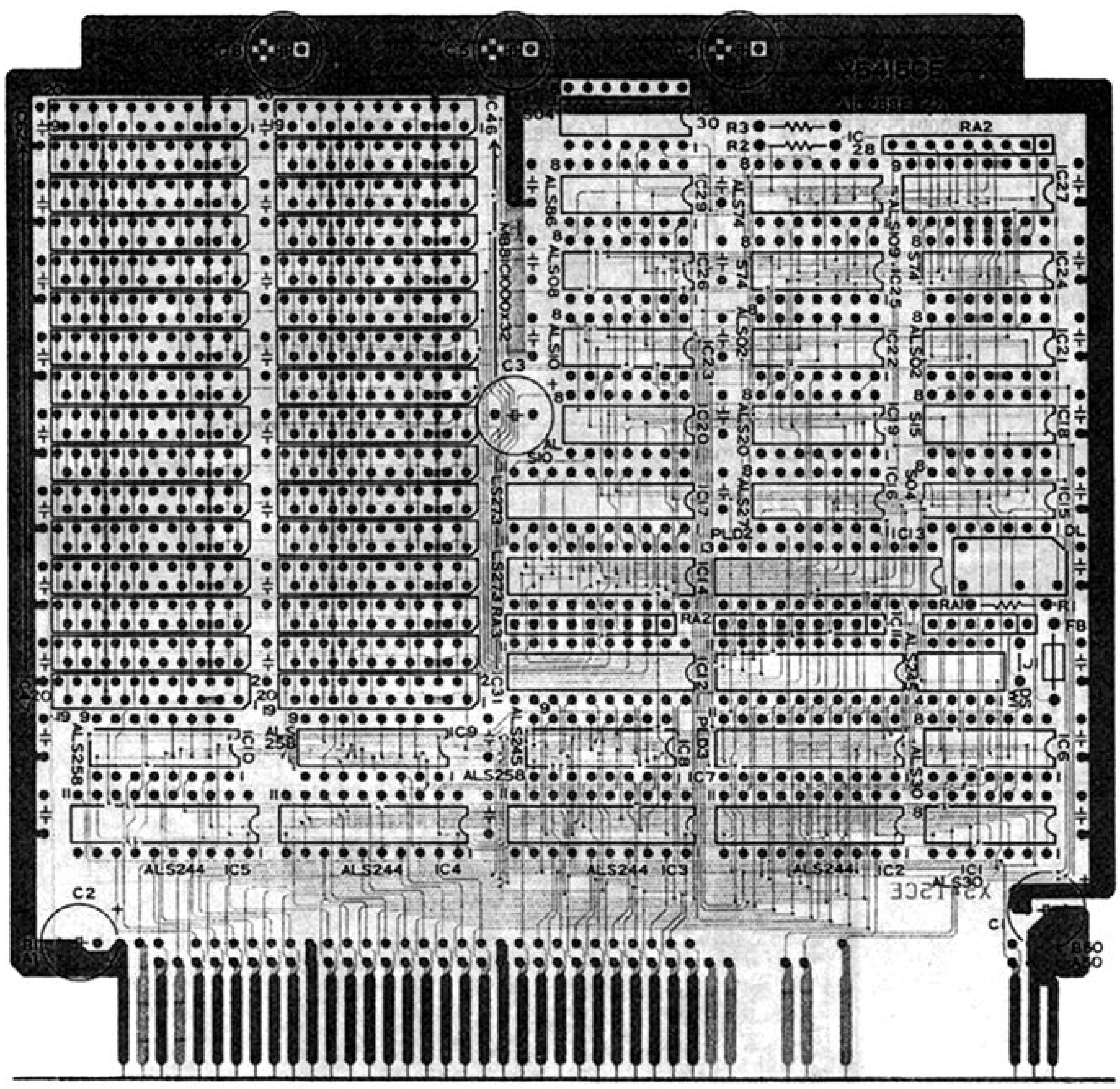
●図……4　6BE2B の部品配置



●図……5　6BE2D の部品配置



●図……6　CZ-6BE2（6BE4）の部品配置

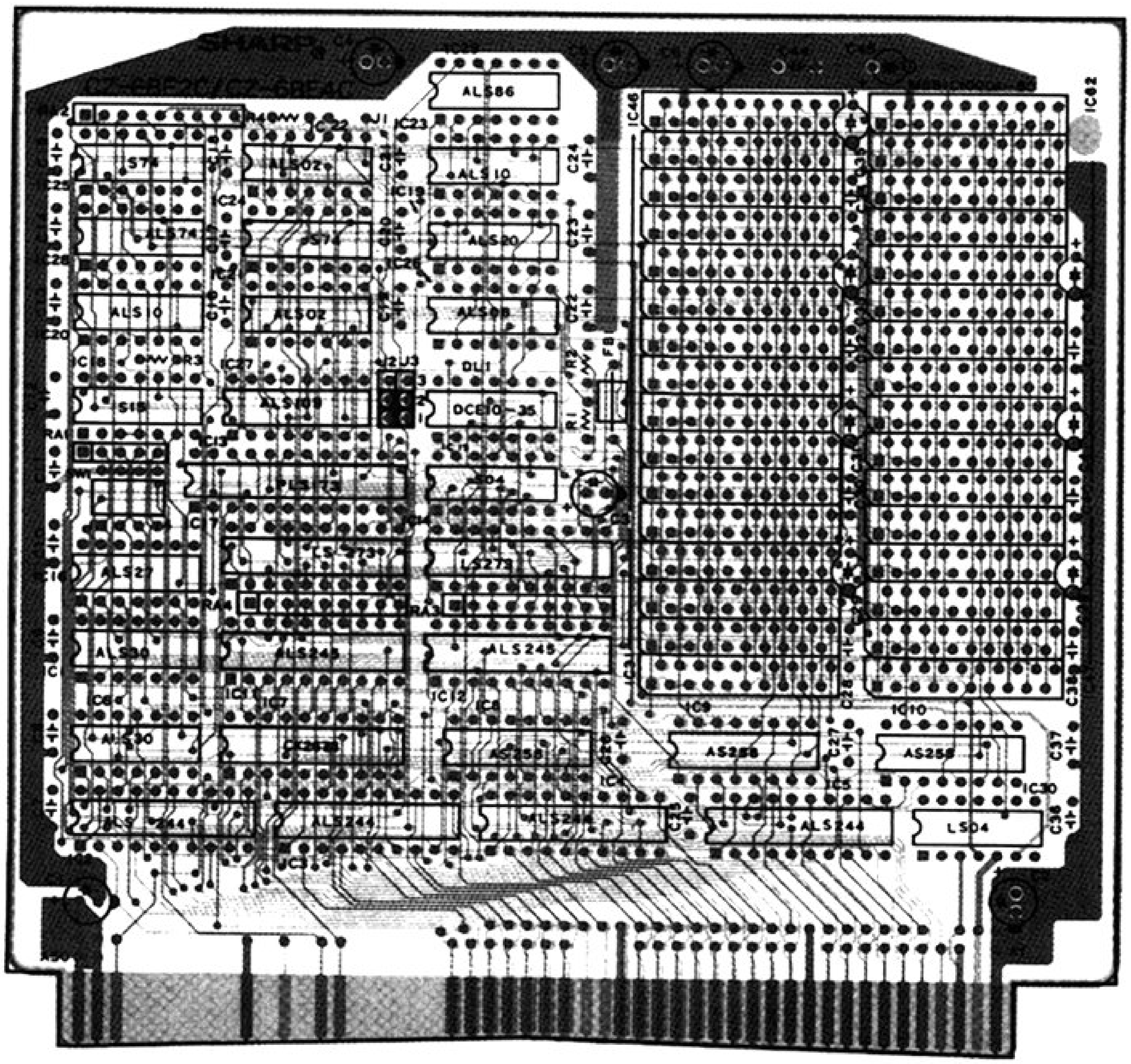


●3 回路図/コネクタ信号配置

❸·1 内部増設用

CZ-6BE1 と 6BE1A (6BE1A (A)) の回路図を図 8 と図 9 に，CZ-6BE2A と CZ-6BE2D の回路図，およびコネクタ信号配置を図 10，図 11，図 12 に示します。CZ-6BE2A/6BE2D 用の増設メモリモジュール，CZ-6BE2B は共通ですので，コネクタの信号も同じものとなっています。

●図……7　6BE2C（6BE4C）の部品配置



●図……8　CZ-6BE1 の回路図（巻末参照）

●図……9　6BE1A（6BE1A(A)）の回路図（巻末参照）

●図……10　CZ-6BE2A の回路図（巻末参照）

●図……11　CZ-6BE2D の回路図（巻末参照）

❸·2　外部拡張用

CZ-6BE2(6BE4)/6BE2C（6BE4C）の回路図を図 13，図 14 に示します。

●図……12　拡張メモリボードのコネクタ信号配置

端子信号	信号名	機　能	端子信号	信号名	機　能
TB1-A1	GND	グランド	TB2-A1	MD15	データ
TB1-A2	GND	グランド	TB2-A2	MD14	データ
TB1-A3	GND	グランド	TB2-A3	MD7	データ
TB1-A4	$\overline{\text{RAS1}}$	ロウアドレスストローブ1	TB2-A4	MD5	データ
TB1-A5	Vcc1	+5V	TB2-A5	MD7	メモリアドレス
TB1-A6	Vcc1	+5V	TB2-A6	MD4	データ
TB1-A7	Vcc1	+5V	TB2-A7	MD2	データ
TB1-A8		N.C.	TB2-A8	MD0	データ
TB1-B1	$\overline{\text{DRAMSENSE3}}$	メモリボード 有(0) 無(1)(CZ6BE2B)	TB2-B1	MD12	データ
		メモリアドレス 600000H～7FFFFF H	TB2-B2	MD11	データ
TB1-B2	$\overline{\text{DRAMSENSE1}}$	メモリボード 有(0) 無(1)(CZ6BE2A)	TB2-B3	MD6	データ
		メモリアドレス 200000H～3FFFFF H	TB2-B4	MA9	メモリアドレス
TB1-B3		N.C	TB2-B5	MA5	メモリアドレス
TB1-B4	$\overline{\text{RAS2}}$	ロウアドレスストローブ2	TB2-B6	MD3	データ
TB1-B5	$\overline{\text{RAS3}}$	ロウアドレスストローブ3	TB2-B7	MD1	データ
TB1-B6	$\overline{\text{CASL1}}$	下位カラムアドレスストローブ1	TB2-C1	MD13	データ
TB1-B7	$\overline{\text{CASU1}}$	上位カラムアドレスストローブ1	TB2-C2	MD10	データ
TB1-C1	$\overline{\text{SOKET OE}}$	ROM用コントロール信号(Output Enable)	TB2-C3	MD8	データ
TB1-C2	$\overline{\text{DRAMSENSE2}}$	メモリボード 有(0) 無(1)(CZ6BE2B)	TB2-C4	MD9	データ
		メモリアドレス 400000H～5FFFFF H	TB2-C5	MA8	メモリアドレス
TB1-C3	$\overline{\text{OE1}}$	メモリコントロール信号(Output Enable)	TB2-C6	MA4	メモリアドレス
TB1-C4	MA17	メモリアドレス	TB2-C7	MA3	メモリアドレス
TB1-C5	MA15	メモリアドレス	TB2-C8	MA2	メモリアドレス
TB1-C6	MA14	メモリアドレス	TB2-D1	GND	グランド
TB1-C7	MA10	メモリアドレス	TB2-D2	GND	グランド
TB1-C8		N.C	TB2-D3	GND	グランド
TB1-D1	ROMSEL	本体4MマスクROM/メモリボード1MEPROM	TB2-D4	MA6	メモリアドレス
		切換信号	TB2-D5	Vcc1	+5V
TB1-D2	$\overline{\text{WE1}}$	メモリ書込信号	TB2-D6	Vcc1	+5V
TB1-D3	MA11	メモリアドレス	TB2-D7	Vcc1	+5V
TB1-D4	MA13	メモリアドレス			
TB1-D5	MA16	メモリアドレス			
TB1-D6	MA12	メモリアドレス			
TB1-D7	MA1	メモリアドレス			

●図……13　CZ-6BE2(6BE4)の回路図（巻末参照）

●図……14　CZ-6BE2C(6BE4C)の回路図（巻末参照）

数値演算プロセッサボード (CZ-6BP1/CZ-6BP1A)

実数演算性能を大幅に向上させる数値演算プロセッサボードを使うと一日がかりだった計算が一時間もかからなくなるのも珍しくありません。三次元グラフィックスやレイ・トレーシングなどの用途では必須と呼んでもよいボードでしょう。

数値演算プロセッサボードは，数値演算（特に浮動小数点演算）を高速に処理することができるLSI, MC 68881を搭載したボードです。X68000の数値演算能力を大幅に向上させられるため，レイ・トレーシングなど数値演算を多用するアプリケーションで効果を発揮します。

1 仕様

主要LSI	
浮動小数点プロセッサ	MC 68881
クロック周波数	16.6 MHz
浮動小数点演算仕様	IEEE std P 754 準拠
電源	
電源電圧	+5 V
消費電流	約 250 mA
動作温度範囲	10～35℃

ポートアドレス	$E9E000〜$E9E 01F/$E9E080〜$E9E09F (ジャンパースイッチで選択)
割り込み	使用せず

●2 回路概要

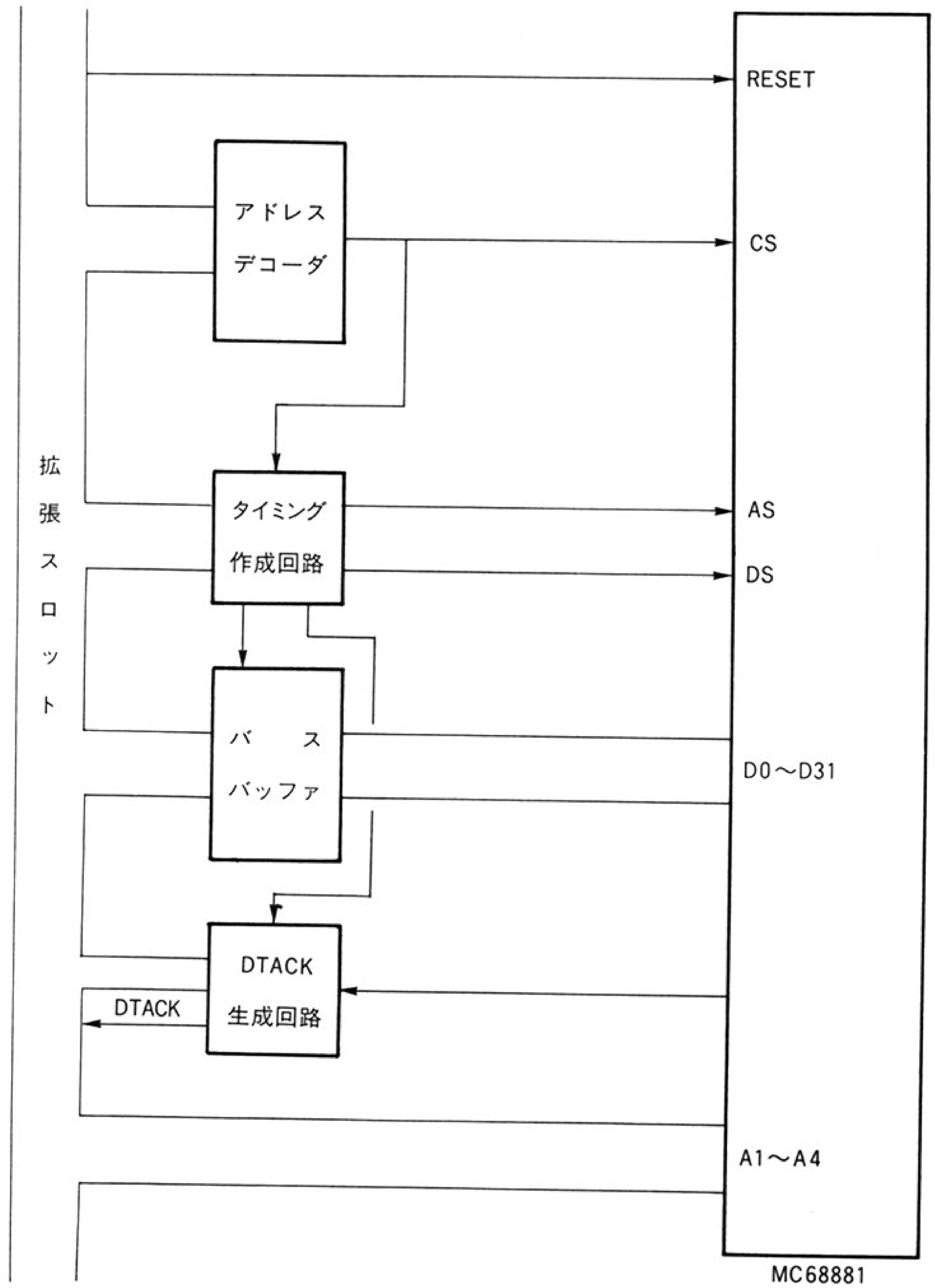
CZ-6BP1/CZ-6BP1Aのブロック図を図1に示します。68000の場合，CPU自身にはコプロセッサと直接インタフェースする機能はありません。このため，CZ-6BP1/CZ-6BP1 Aでは MC 68881をI/Oデバイスとして接続するようになっています。

MC 68881はI/Oデバイスとして接続されるため，X68000側と同じクロックで動作する必要はありません。CZ-6BP1/CZ-6BP1AではMC 68881をチップの最高速度である16.665 MHzのクロックで動作させています。

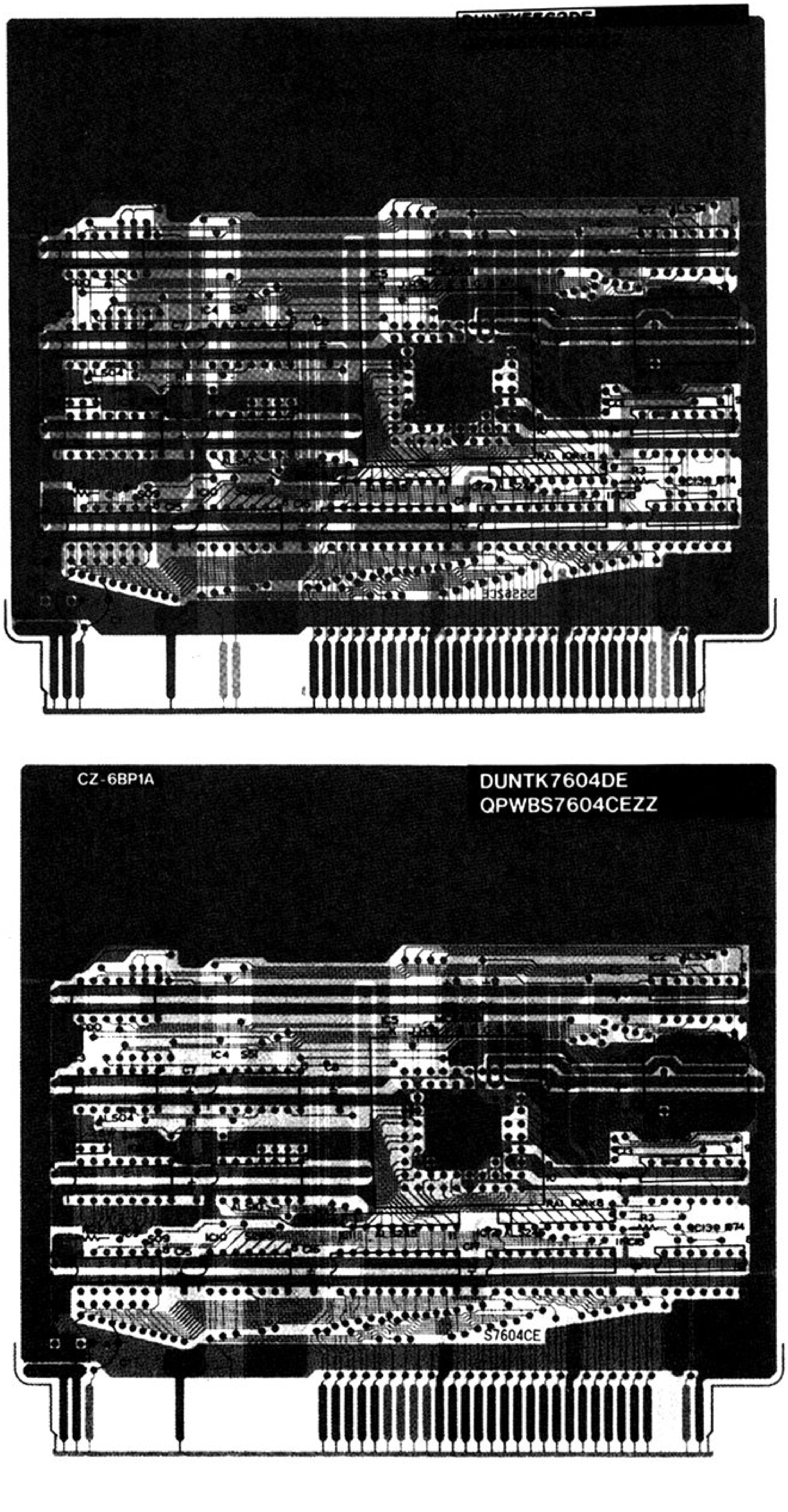
●3 主要部品配置

CZ-6BP1/CZ-6BP1Aの部品の配置を図2に示します。

●図……1　数値演算プロセッサボードのブロック図



●図……2　CZ-6BP1/CZ-6BP1A の部品配置

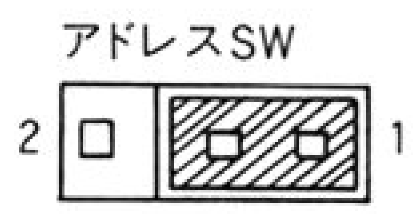


●4 設　定

❹·1 ポートアドレスの設定

CZ-6BP1/CZ-6BP1A は，ボード上のジャンパースイッチ（SW 1）によってポートアドレスを切り替えられるようになっています(図3)。アドレススイッチが1側のときポートアドレスは$E9E000～$E9E01F に，2側のとき$E9E080～$E9E09F になります。

●図……3　ジャンパースイッチ (SW 1)



工場出荷時は〝1〟側に設定されています。

アドレスSW	ポートアドレス
1　側	$E9E000 ～ $E9E020
2　側	$E9E080 ～ $E9E0A0

現在シャープから提供されている浮動小数点ドライバは1側に設定したボードしかサポートしていませんので，2に設定した場合はドライバを改造したり，直接コプロセッサを操作するようなプログラムを書いて使うことになります。

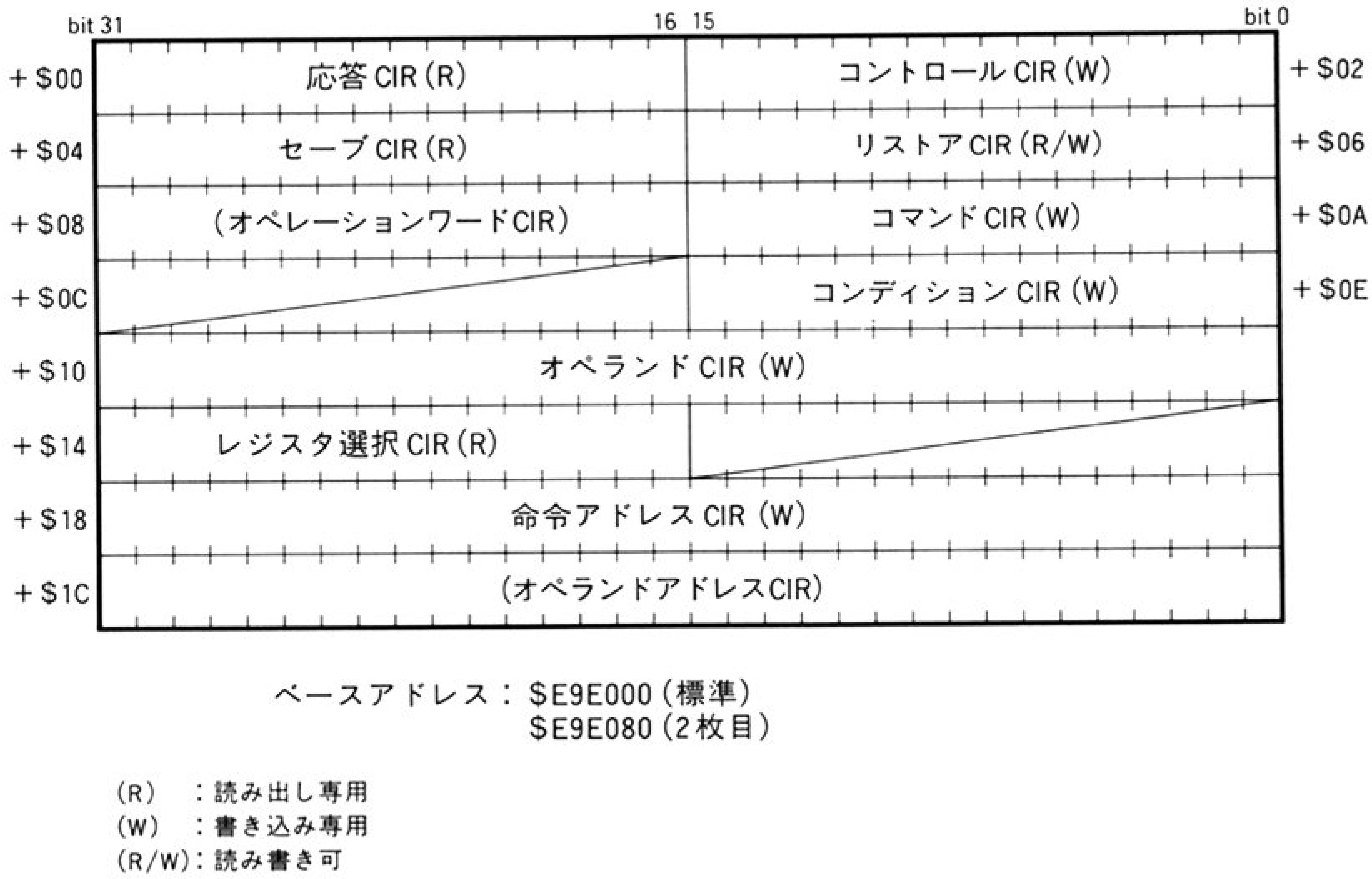
●5 I/Oマップ

MC 68881 は内部に8本の浮動小数点レジスタやステータスレジスタなどを持っており，実際の演算処理はこれらのレジスタを使用して行われます。ただし CPU からはこれらのレジスタに直接アクセスできるわけではなく，CIR（コプロセッサ・インタフェース・レジスタ）と呼ばれるレジスタ群を用いて間接的にアクセスするようになっていることに気をつける必要があ

ります。

CZ-6BP1/CZ-6BP1A の I/O アドレスマップを図 4 に示します。すでに述べたとおり，ベースアドレスは$E9E 000 と$E9E080 のいずれかを選択することができるようになっています。MC 68881 の具体的な使用方法については拙著『Inside X68000』で実例とともに説明しておりますので，参考にしてください。

●図……4　数値演算プロセッサボードの I/O アドレスマップ



●6 回路図

CZ-6BP1/CZ-6BP1A の回路図を図 5 と図 6 に示します。

●図……5　CZ-6BP1 の回路図 (巻末参照)

●図……6　CZ-6BP1A の回路図 (巻末参照)

MIDIボード
(CZ-6BM1)

安価な音源の発売と共にDTM(デスクトップミュージック)の世界の主役となったMIDIはもともとプロのミュージシャンの世界のものでした。MIDIボードはこのMIDIの世界とX68000をつなぐインタフェースボードです。

MIDI ボード(CZ-6BM1)は，シンセサイザーやシーケンサ，リズムマシンなどを相互に接続するための規格となっているシリアル通信インタフェース，MIDI (Music Instrument Device Interface) をサポートするボードです(実際に音楽を楽しむためには本ボードのほかにMIDI楽器が必要です)。

CZ-6BM1はMIDI OUT端子を2つ，MIDI IN端子を1つ持っており，さらにMIDI OUT端子のうち1つはMIDI THRU端子として使うこともできるようになっています。また，テープシンク入出力端子を持っており，マルチトラックレコーダによる同期録音，同期演奏も可能となっています。

1 仕様

主要LSI

MIDIコントローラ(MCS)	YM 3802
クロック周波数	4 MHz

MIDI バス	
準拠規格	MIDI 規格 1.0
伝送速度	31.25 Kbps
MIDI 端子	MIDI OUT×2(うち1端子はMIDI THRUと切り替え可)
	MIDI IN ×1
テープ同期信号	FSK 1200 bps
電源	
電源電圧	+5 V
消費電流	約 180 mA
ポートアドレス	$EAFA01～$EAFA0F/$EAFA11～$EAFA1F (ジャンパースイッチで選択)
割り込み	レベル 2/レベル 4 (ジャンパースイッチで選択)

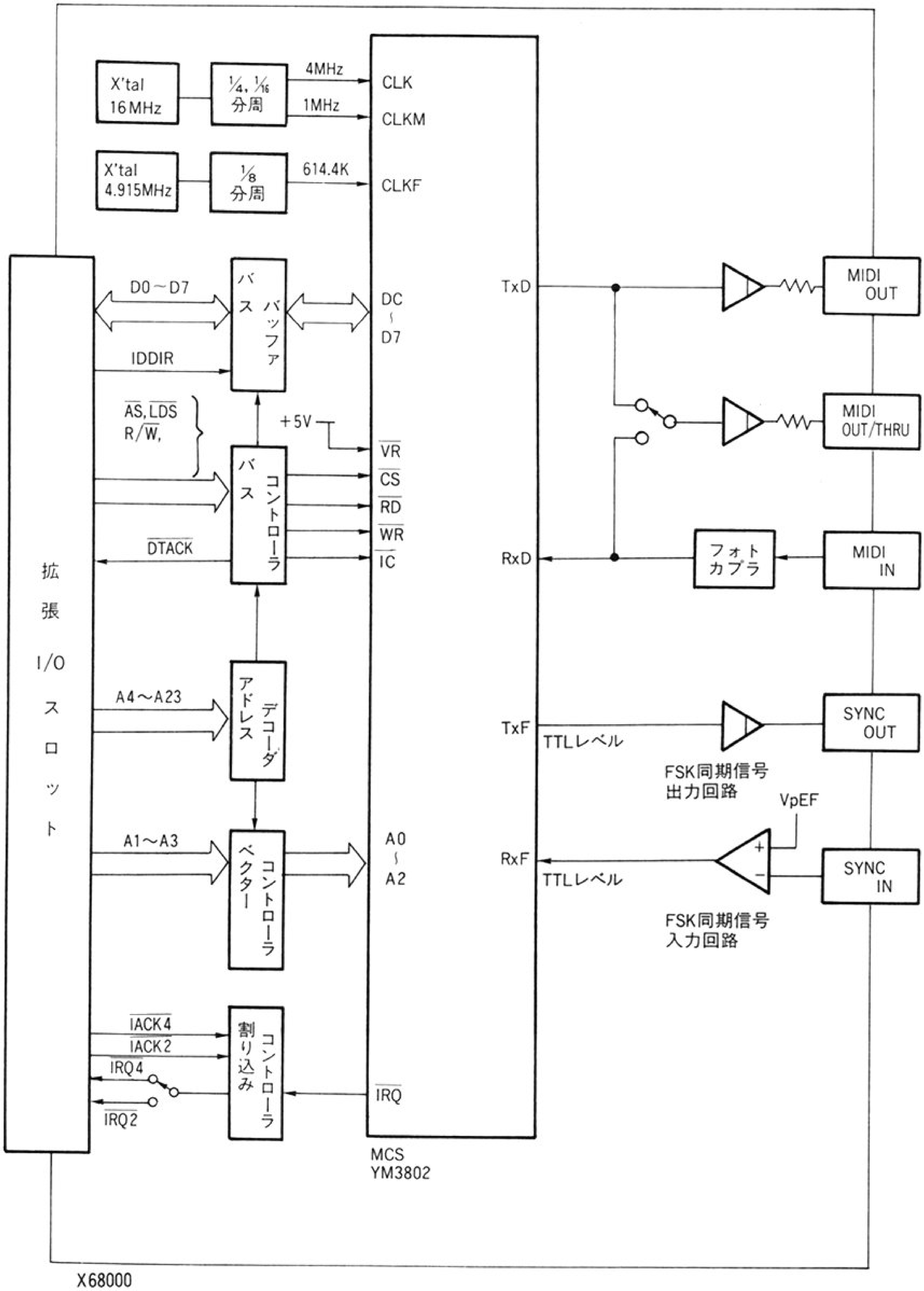
●2 回路概要

CZ-6BM1 のブロック図を図 1 に示します。MIDI コントロール LSI としてヤマハの YM3802 が使用されています。YM3802 は 3 種類のクロックを必要とします。CZ-6BM1 では 16 MHz の原発振を 1/4 に分周して 4 MHz のシステムクロックに，1/16 に分周して得た 1 MHz をボーレート生成用のクロックに使用します。さらに，4.9152 MHz の発振器の出力を 1/8 に分周して得た 614.4 KHz を MIDI のアクティブセンスの計時用クロックとして CLKF 端子に入れています。

MIDI 入力はフォトカプラで受信され，出力はオープンコレクタのバッファで行われています。MIDI 出力のうち片側はジャンパースイッチでもう一方の MIDI OUT と同一の出力とするか，MIDI THRU (MIDI IN からの入力をそのまま出力する) にするかを選択できるようになっています。

SYNC OUT 出力からは MIDI クロックを 1200 bps で FSK 変調した信号が出力されます。SYNC IN から入力された信号はダイオードによるリミッターによって振幅が抑えられた後，コンパレータで波形整形されて YM3802 に入力されるようになっています。

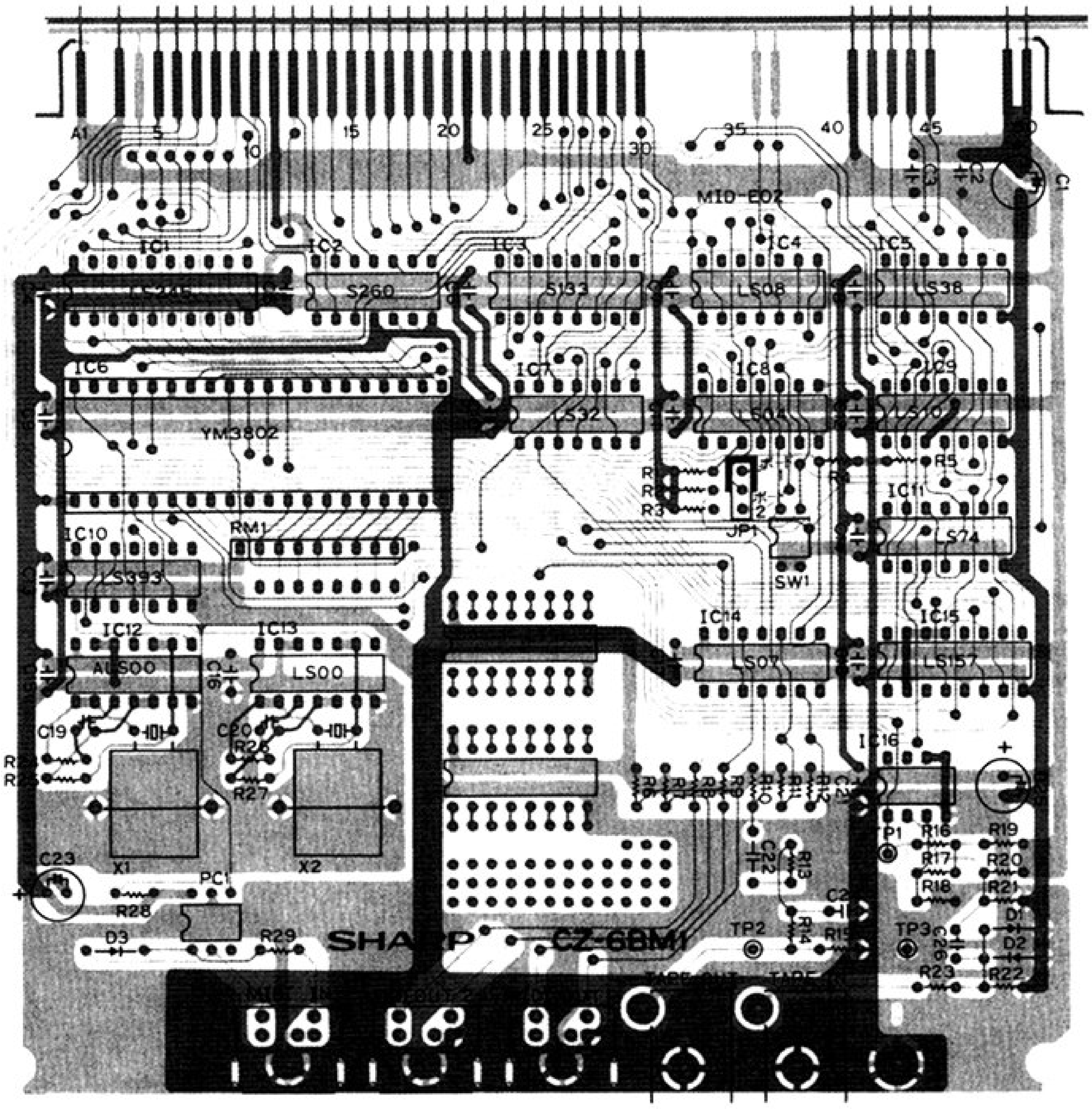
●図……1　MIDI ボードのブロック図



●3 主要部品配置

CZ-6BM1の部品配置を図2に示します。

●図……2　CZ-6BM1の部品配置



●4 設 定

❹·1 ポートアドレスの設定

CZ-6BM1 はジャンパースイッチ (JP1) によって，ポートアドレスを選択できるようになっています。1 側になっていると$EAFA01～$EAFA0F に，2 側になっていると$EAFA11～$EAFA1F になります。工場出荷時は 1 側に設定されています。

❹·2 割り込みレベルの設定

ボード上のディップスイッチ 1 の 1 側(スイッチ 1-1) でボードが使用する割り込みレベルを選択できるようになっています。ON のとき割り込みレベル 4 が，OFF のとき割り込みレベル 2 が使用されます。

❹·3 MIDI OUT/MIDI THRUの選択

ボード用のディップスイッチ 1 の 2 側 (スイッチ 1-2) で MIDI OUT/MIDI THRU 兼用端子を MIDI OUT で使用するか，MIDI THRU として使用するかの選択を行うようになっています。ON のとき MIDI OUT に，OFF のとき MIDI THRU として動作します。

●5 I/Oマップ

CZ-6BM1 の I/O アドレスマップを図 3 に示します。YM3802 には 38 本のレジスタがあるためそれらをグループ分けし，占有する I/O アドレスの削減を図っています。

●図……3　MIDI ボードの I/O アドレスマップ

デバイス	アドレス	アクセスされるレジスタ
	ベースアドレス ＋$01	R00
YM3802	＋$03	R01
	＋$05	R02
	＋$07	R03
	＋$09	R04, R14, R24, R34, R44, R54, R64, R74, R84, R94
	＋$0B	R05, R15, R25, R35, R45, R55, R65, R75, R85, R95
	＋$0D	R06, R16, R26, R36, R56, R66, R76, R86, R96
	＋$0F	R17, R27, R67, R77, R87

ベースアドレス：$EAFA00（1 枚目）/ $EAFA10（2 枚目）

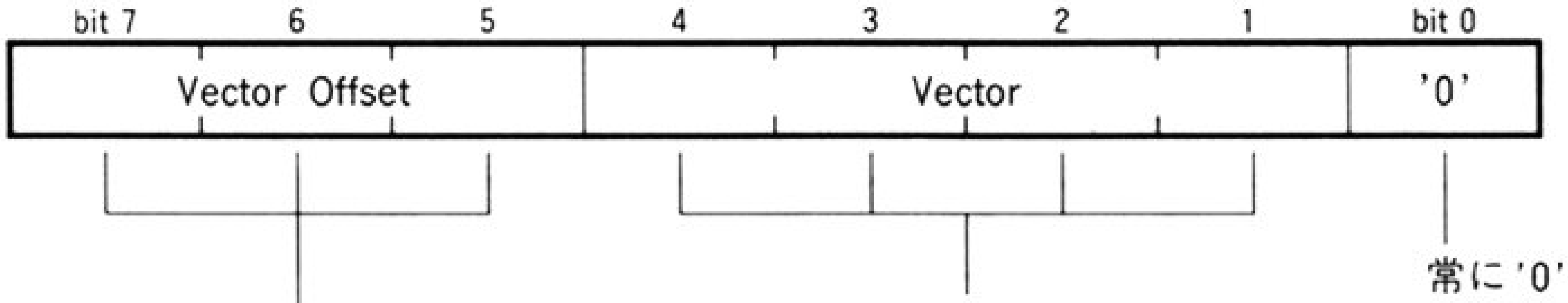
R00 から R03 までの 4 本のレジスタは，それぞれ$EAFA01/$EAFA11 から$EAFA07/$EAFA17 に配置されており，常時アクセスが可能です。

その他のレジスタはまず R01 にレジスタ番号の上位の値（たとえば R45 なら 4）を書き込んでから下位の値（R45 なら 5）に対応するポートアドレス（R45 なら$EAFA0B/$EAFA1B）にアクセスするようになっています。つまり，レジスタ番号の下位が同じ値になっているレジスタは，同じポートアドレスを共有しており，それらのうち R01 に書き込んだ値がレジスタ番号の上位と一致するものがアクセスされるようになっているわけです。

YM 3802 ではレジスタ番号の上位データをグループ番号と呼んでいます。R01 に書き込んだ値はレジスタアクセス後も変化しませんので，グループ番号が同じレジスタへのアクセスであれば，R01 への再書き込みは省略してもかまいません。

YM3802 の各レジスタのビット配置を図 4 ～図 39 に示しますので，参考にしてください。

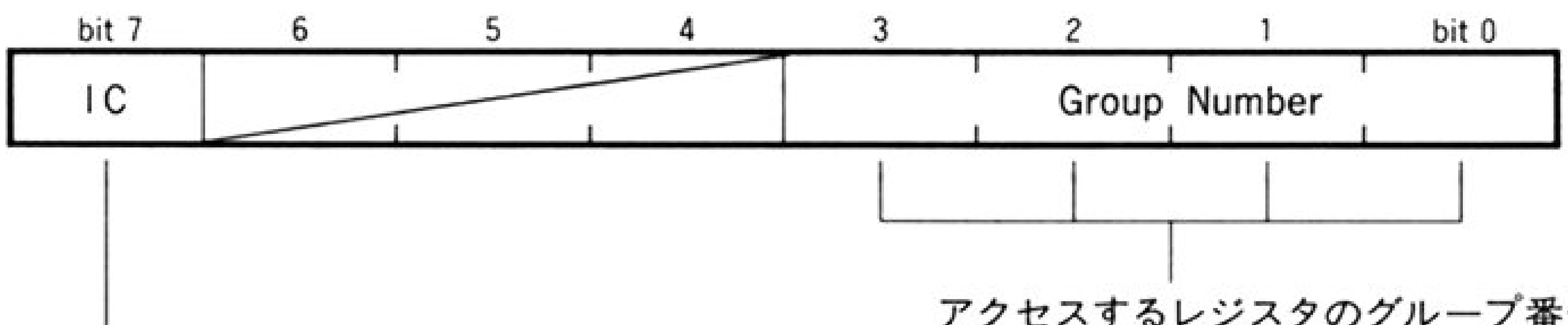
●図……4　R00（Read）：割り込みベクタ



ベクタの上位ビット
（R04の上位3ビットと同一）

割り込み発生要因を示す
'1000'：現在アクティブな割り込み要因はない
'0111'：General Timer（タイマのカウント値が0になった）
'0110'：FIFO-Tx empty（送信FIFOが空になった）
'0101'：FIFO-Rx ready（受信FIFOにデータがセットされた）
'0100'：Off-line detected（300mSの間受信が行われなかった）
　　　　/Break detected（2キャラクタの間シリアル入力が'L'）
'0011'：Recording Counter（レコーディングカウンタが0になった）
'0010'：Play-back Counter（プレイバックカウンタが負になった）
'0001'：Click Counter（カウント値が0になった）
　　　　/MIDI-clock detected（FIFO-IRxの最古のデータが$F8）
'0000'：MIDI real-time message detected
　　　　（FIFO-IRxの最古データが$F9～$FD）

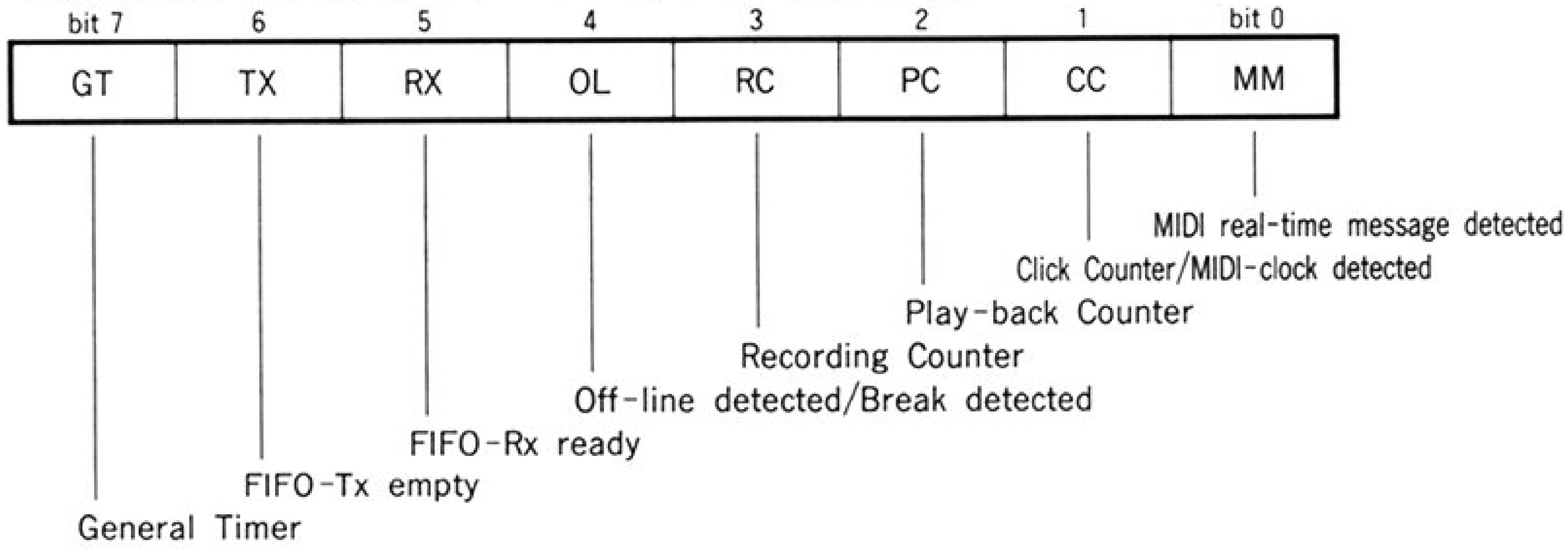
●図……5　R01（Write）：システムステータス



Initial Clear Request
32クロック以上'1'にすると，YM3802がリセットされる
再度使用する場合には'0'に戻すこと

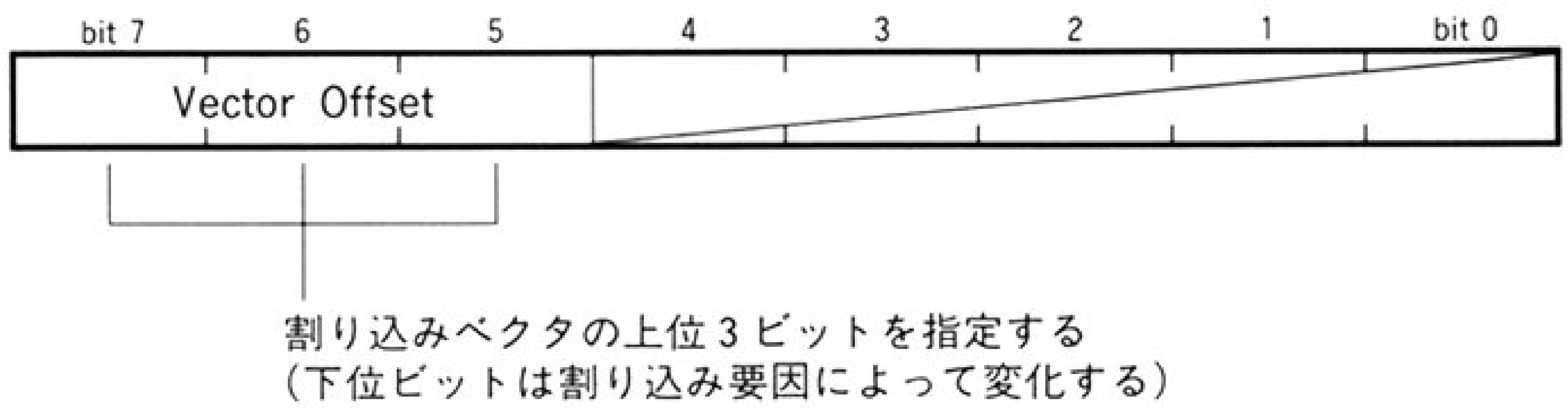
●図……6　R02 (Read),R03 (Write),R06 (Write)

- R02(Read)：割り込みステータス（'1'：要求あり）
- R03(Write)：割り込みクリア　（'1'：クリア）
- R06(Write)：割り込みイネーブル（'1'：割り込み許可）

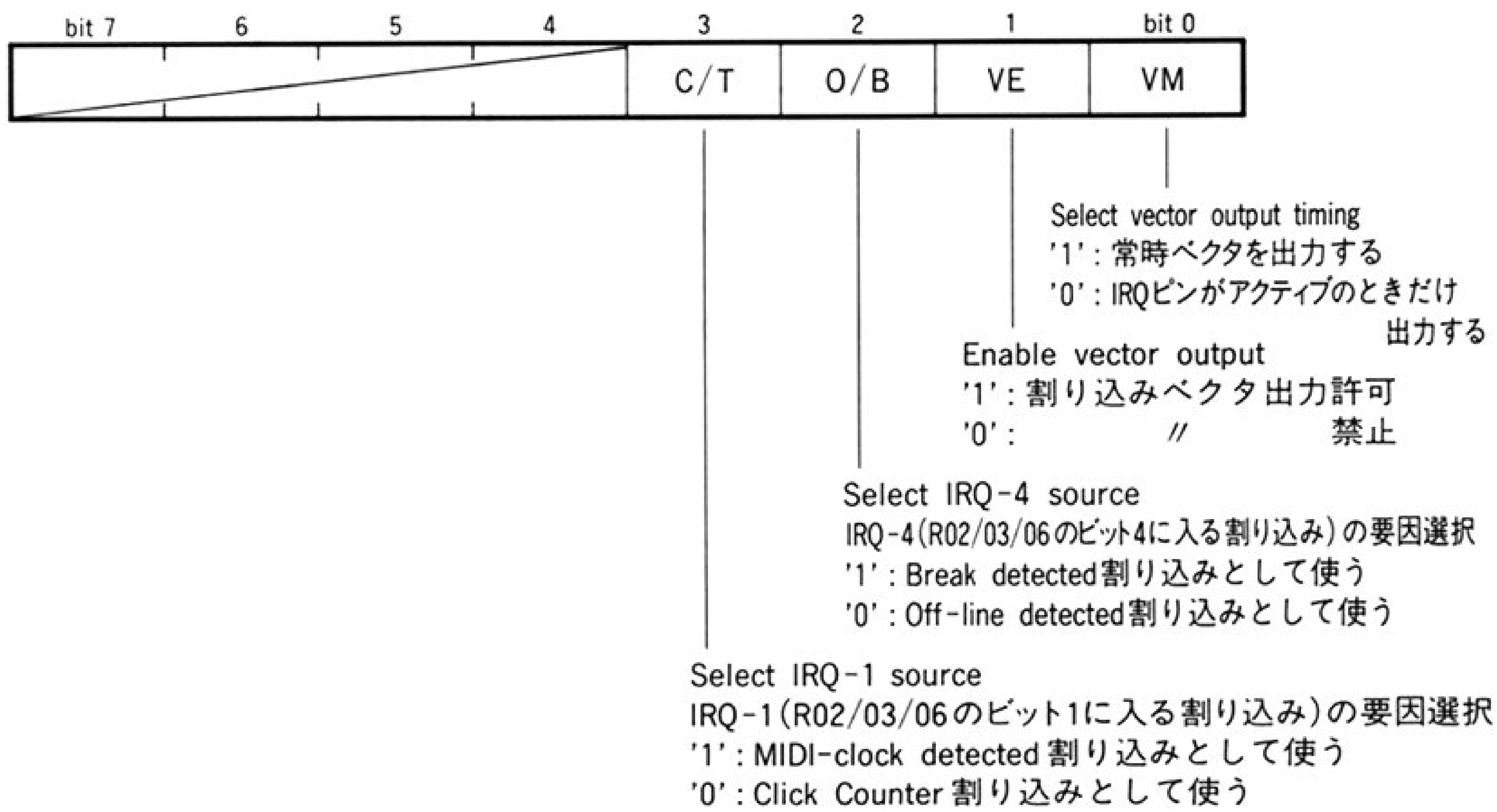


※各要因の内容は図4のR00を参照してください

●図……7　R04 (Write)：割り込みベクタオフセット



●図……8　R05 (Write)：割り込みモードコントロール



●図……9　R14 (Write)：MIDI リアルタイムメッセージ制御

bit 7	6	5	4	3	2	1	bit 0
		ASE	MCE	CDE	MCDS	MCFS	

Select MIDI-clock for FIFO-IRx
(FIFO-Rxに対するクロックメッセージのセットがいずれの要因によって行われるかを選択する)
'11' : MIDI-clock Timer
'11' : SYNC Detector
'00' : MIDI message Detector
'00' : (Inhibited)

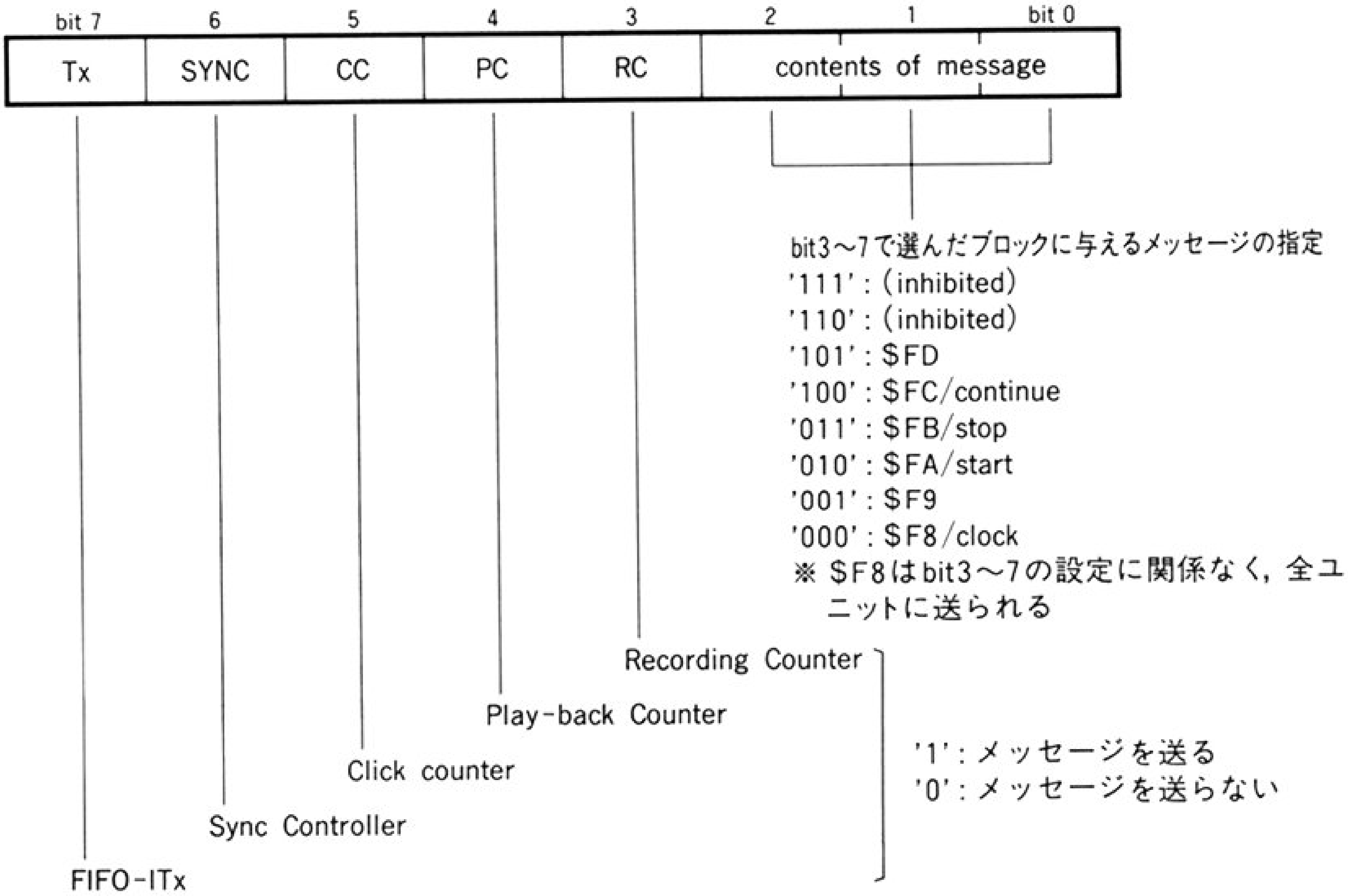
Select MIDI-clock for distributer
(MIDIメッセージディテクタの内部MIDIクロック発振源選択)
'1' : user (R15の操作)
'0' : FIFO-IRx

Enable MIDI-control ($F9～$FD) detection
($F9～$FDのシステムリアルタイムメッセージの受信制御)
'1' : 許可
'0' : 禁止

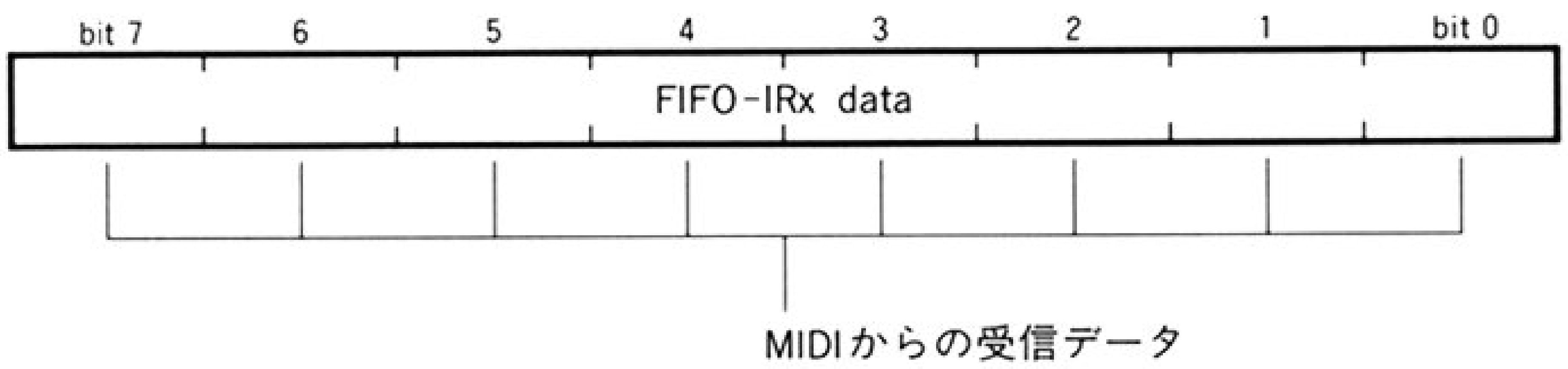
Enable auto MIDI-clock ($F8) output
(MIDIメッセージディストリビュータからのクロックメッセージ送信制御)
'1' : 許可
'0' : 禁止

Enable auto active-sense ($FE) output
(アイドルディテクタによるアクティブセンスメッセージ送信制御)
'1' : 許可
'0' : 禁止

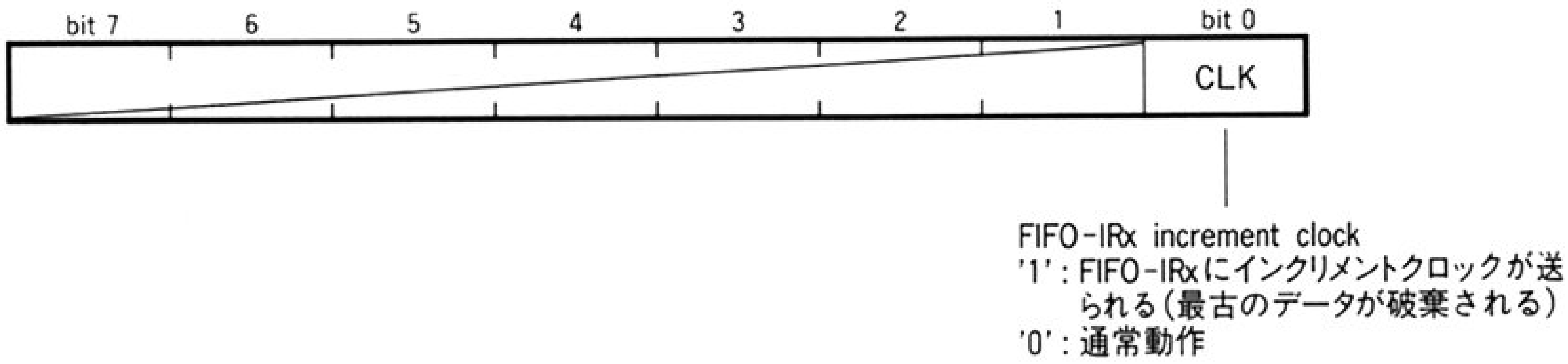
●図……10　R15 (Write)：MIDI リアルタイムメッセージ要求



●図……11　R16 (Read)：FIFO-IRx data



●図……12　R17 (Write)：FIFO-IRx コントロール



●図……13　R24 (Write)：Rx コミュニケーションレート

bit 7	6	5	4	3	2	1	bit 0
		RxD/F	communication rate				

communication rate (bit 4–0):
MIDI の受信速度を決める
'00XXX' : CLKM/16　(RxD/F='1'のときは禁止)
'01XXX' : CLKM/32　(　　〃　　)
'10XXX' : CLKF/32
'11000' : CLKF/64
'11001' : CLKF/128
'11010' : CLKF/256
'11011' : CLKF/512
'11100' : CLKF/1024
'11101' : CLKF/2048
'11110' : CLKF/4096
'11111' : CLKF/8192

RxD/F (bit 5):
Select Receiver input connection
レシーバの入力をどこから取るかを決める
'1' : FSK demodulator
'0' : RxD

●図……14　R25 (Write)：Rx コミュニケーションモード

bit 7	6	5	4	3	2	1	bit 0
		RxCL	RxPE	RxPL	RxE/O	RxSL	RxST

RxST (bit 0):
Select stop-bit type in 2bit length
'1' : LH
'0' : HH

RxSL (bit 1):
Select stop bit length
'1' : 2bit
'0' : 1bit

RxE/O (bit 2):
Select parity-bit polarity
'1' : odd
'0' : even

RxPL (bit 3):
Select parity-bit length
'1' : 4bit
'0' : 1bit

RxPE (bit 4):
Enable parity-bit check
'1' : enable
'0' : disable

RxCL (bit 5):
Select data bit length
'1' : 7bit
'0' : 8bit

●図……15　R26 (Write)：Address-hunter コントロール(1)

bit 7	6	5	4	3	2	1	bit 0
IDCL	define maker-ID code						

bit 6〜0：メーカのIDコードを設定する

bit 7：デバイスIDコードのチェックを行うか否かを指定する
'1'：Maker-ID & device-ID
'0'：Maker-ID only

●図……16　R27 (Write)：Address-hunter コントロール(2)

bit 7	6	5	4	3	2	1	bit 0
BDRE	define device-ID code						

bit 6〜0：デバイスIDコードを設定する

bit 7：アドレスが$7Fのデータを受け取るか否かを決める
'1'：設定されたアドレスのほか、$7Fも取り込む
'0'：設定されたアドレス以外のデータは受け取らない

●図……17　R34 (Write)：FIFO-Rx ステータス

bit 7	6	5	4	3	2	1	bit 0
RxRDY	RxOV	RxF	RxP	BRK	RxOL	AHBSY	RxBSY

bit 0 (RxBSY)：Receiver busy flag
'1'：Receiver busy
'0'：Receiver idle

bit 1 (AHBSY)：Address-hunter busy flag
'1'：Address-hunter busy
（アドレスハンタが動作中にMIDIのシステムエクスクルーシブメッセージを検出した）
'0'：Address-hunter idle

bit 2 (RxOL)：Off-line detected flag
'1'：オフライン状態が検出された
'0'：通常動作

bit 3 (BRK)：Break detected flag
'1'：BREAK状態が検出された
'0'：通常動作

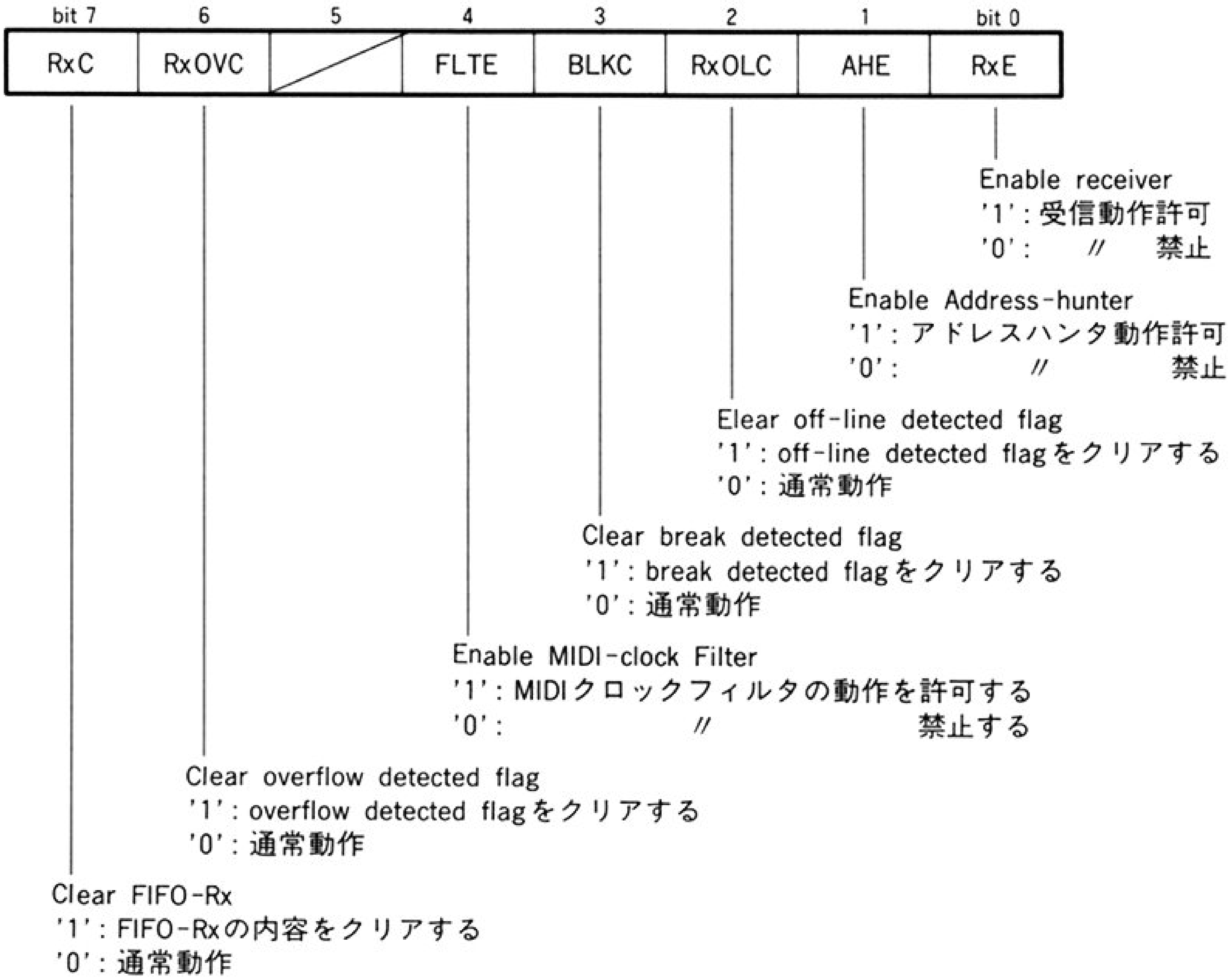
bit 4 (RxP)：Parity error flag
'1'：パリティエラーが検出された
'0'：通常動作

bit 5 (RxF)：Flaming error flag
'1'：フレーミングエラーが検出された
'0'：通常動作

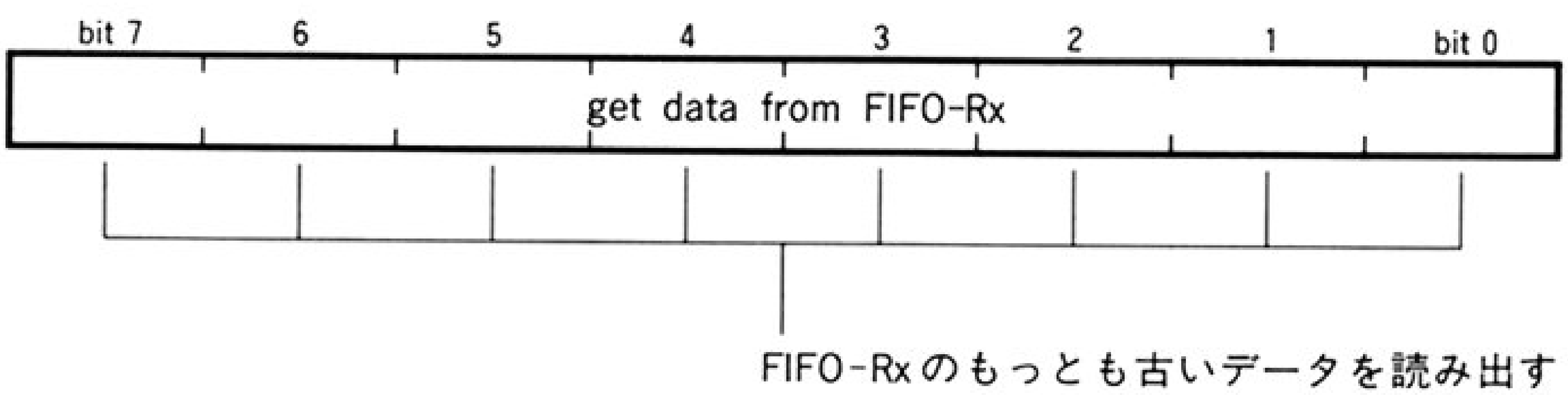
bit 6 (RxOV)：FIFO-Rx overflow detected flag
'1'：オーバーフロー（受信バッファが一杯になった状態で次のデータがきた）が発生した
'0'：通常動作

bit 7 (RxRDY)：FIFO-Rx ready flag
'1'：読み出すべきデータがバッファに入っている
'0'：バッファは空である

●図……18　R35（Write）：FIFO-Rx コントロール



●図……19　R36（Read）：FIFO-Rx data



●図……20　R44 (Write)：Tx コミュニケーションレート

bit 7	6	5	4	3	2	1	bit 0
／	TxRx	TxD/F	Communication rate				

MIDIの送信速度を選択する
'00XXX'：CLKM/16　(TxD/F＝'1'のときは禁止)
'01XXX'：CLKM/32　(　　〃　　)
'10XXX'：CLKF/32
'11000'：CLKF/64
'11001'：CLKF/128
'11010'：CLKF/256
'11011'：CLKF/512
'11100'：CLKF/1024
'11101'：CLKF/2048
'11110'：CLKF/4096
'11111'：CLKF/8192

Select transmitter output connection
送信出力をどこからとるかを決める
'1'：FSK modulator
'0'：TXD

Select TxD connection
'1'：RxD (RxDから受信されたものをそのままTxDから送信する)
'0'：transmitter (通常動作)

●図……21　R45 (Write)：Tx コミュニケーションモード

bit 7	6	5	4	3	2	1	bit 0
／		TxCL	TxPE	TxPL	TxE/O	TxSL	TxST

Select stop-bit type in 2 bit length
'1'：LH
'0'：HH

Select stop-bit length
'1'：2bit
'0'：1bit

Select parity-bit polarity
'1'：odd
'0'：even

Select parity-bit length
'1'：4bit
'0'：1bit

Enable parity-bit generation
'1'：enable
'0'：disable

Select data-bit length
'1'：7bit
'0'：8bit

●図……22　R54（Read）：FIFO-Tx ステータス

bit 7	6	5	4	3	2	1	bit 0
TxEMP	TxRDY				TxIDL		TxBSY

Transmitter busy flag
'1'：トランスミッタ動作中
'0'：トランスミッタアイドル

Tx idle detected flag
'1'：アイドルディテクタによってアイドル状態が検出された
'0'：通常動作

FIFO-Tx ready flag
'1'：送信FIFOに空きがある
'0'：　〃　ない（送信FIFOに書き込んでも無視される）

FIFO-Tx empty flag
'1'：送信FIFOは空である
'0'：　〃　にデータが入っている

●図……23　R55（Write）：FIFO-Tx コントロール

bit 7	6	5	4	3	2	1	bit 0
TxC				BRKE	TxIDLC		TxE

Enable transmitter
'1'：送信動作許可
'0'：　〃　禁止

Clear Tx idle flag
'1'：Tx idleフラグをクリアする
'0'：通常動作

Enable send break
'1'：トランスミッタの出力信号を'L'に固定（ブレーク状態）する
'0'：通常動作

Clear FIFO-Tx & FIFO-ITx
'1'：FIFO-TxとFIFO-ITxの内容をクリアする
'0'：通常動作

●図……24　R56（Write）：FIFO-Tx data

bit 7	6	5	4	3	2	1	bit 0
set data to FIFO-Tx							

送信したいデータを書き込む

●図……25　R64 (Read)：FSK ステータス

bit 7	6	5	4	3	2	1	bit 0
RxFS	SS	CSF	CFF			PS	PDF

Polarity detected flag
'1'：極性検出が行われた
'0'：　　〃　行われていない

Polarity status
'1'：負極性
'0'：正極性

Carrier fast detected flag
'1'：設定されている復調レートよりも速い通信レートの信号が入力されている
'0'：正常動作

Carrier slow detected flag
'1'：設定されている復調レートよりも遅い通信レートの信号が入力されている
'0'：正常動作

Demodulated serial signal status
'1'：mark ('H')
'0'：space ('L')

RxF pin status
'1'：'H' レベル
'0'：'L' レベル

●図……26　R65 (Write)：FSK コントロール

bit 7	6	5	4	3	2	1	bit 0
ME			CFC	DE	APD	P/N	PDFC

Clear polarity detected flag
'1'：polarity detected flag をクリアする
'0'：通常動作

Set polarity by manual
自動極性追従機能を動作させないときの極性指示
'1'：負極性
'0'：正極性

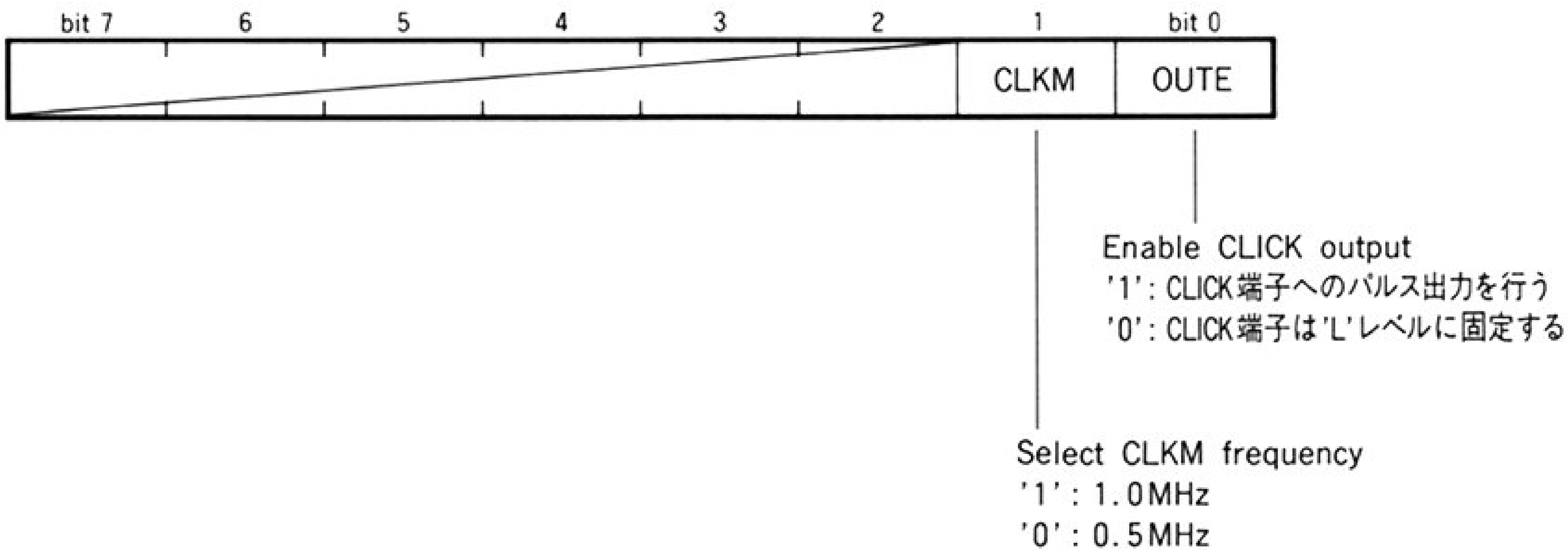
Disable auto polarity detecter
'1'：自動極性追従機能を動作させない
'0'：　　〃　させる

Enable Demodulator
'1'：FSK復調器動作許可
'0'：　〃　禁止

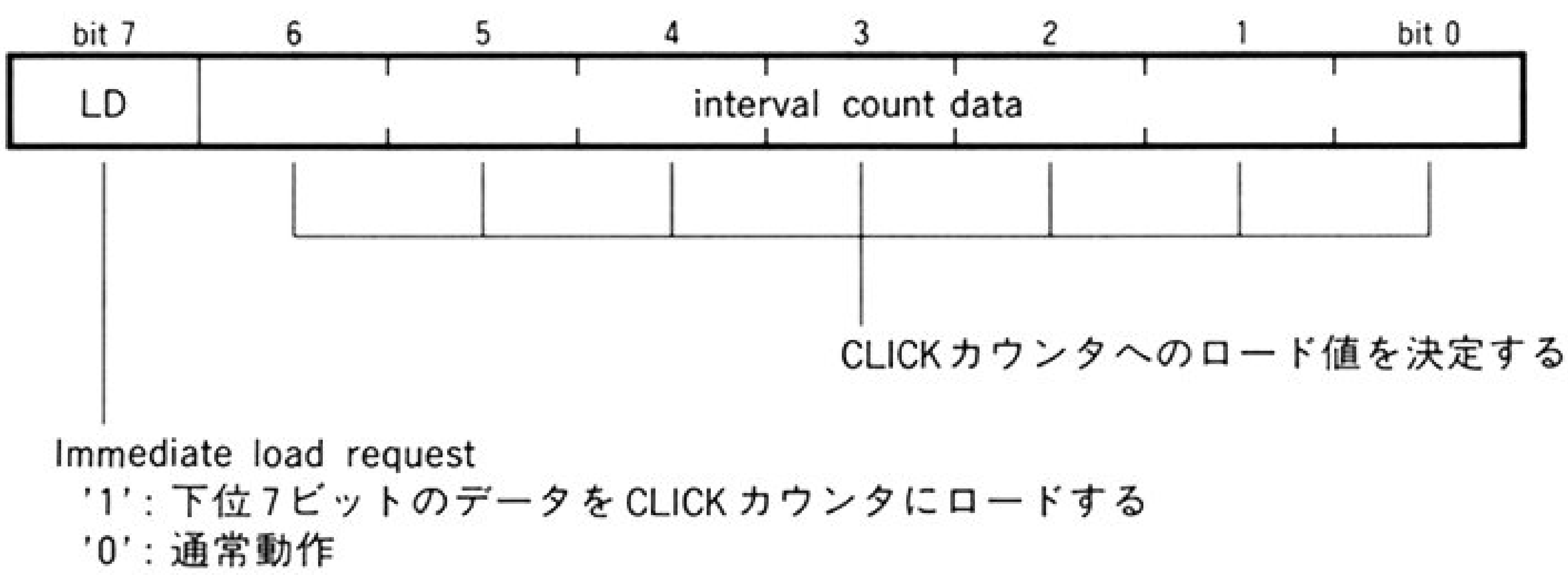
Clear carrier S/F detected flag
'1'：carrier slow/fast detected flag をクリアする
'0'：通常動作

Enable Modulator
'1'：FSK変調器動作許可
'0'：　〃　禁止

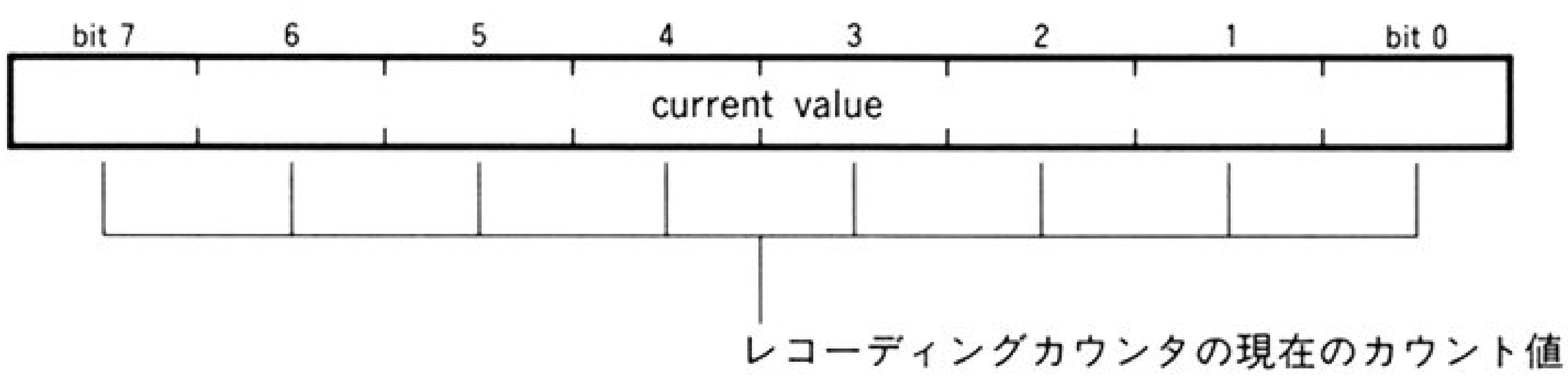
●図……27　R66 (Write) : Click Counter コントロール



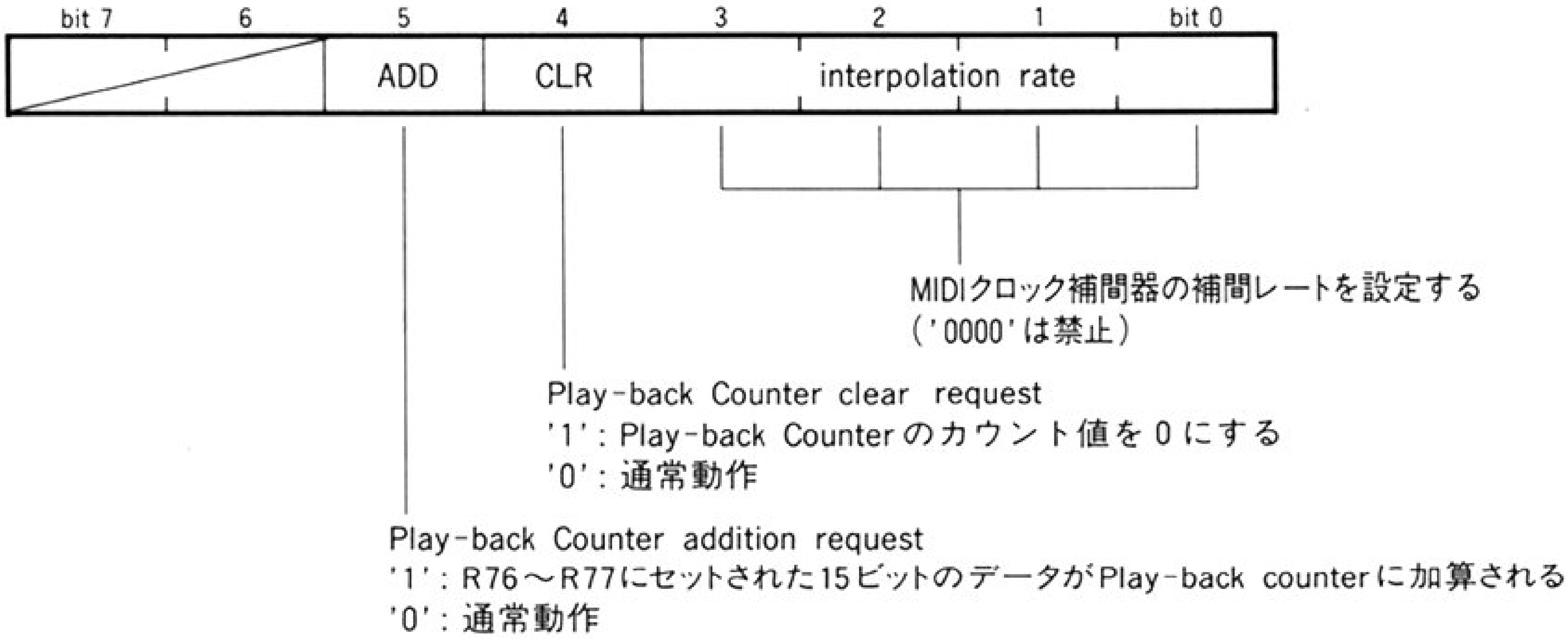
●図……28　R67 (Write) : Click Counter 値



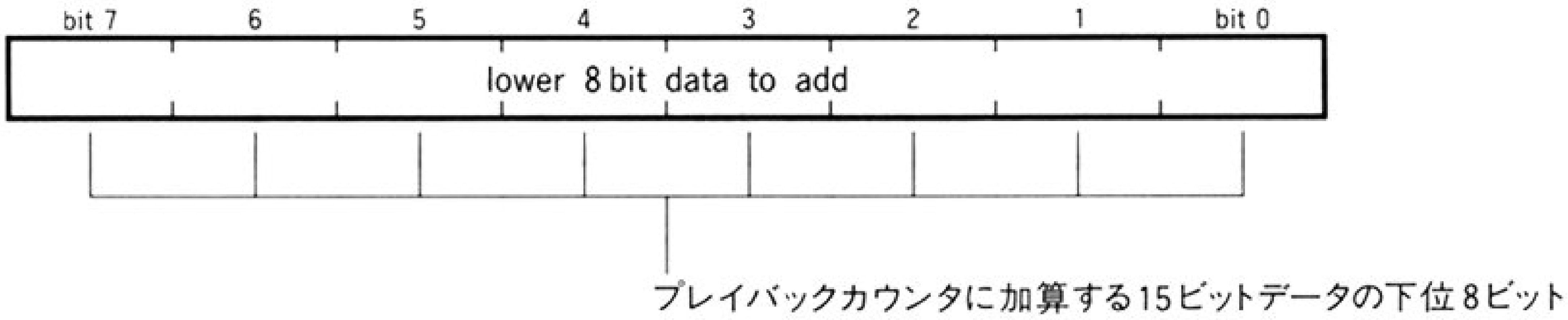
●図……29　R74 (Read) : Recording Counter 現在値



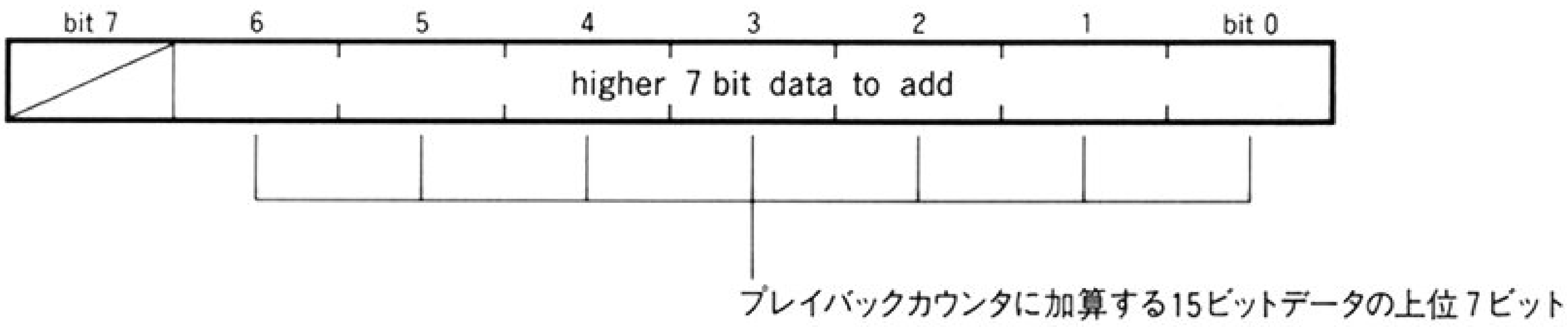
●図……30　R75 (Write) : Interpolator コントロール



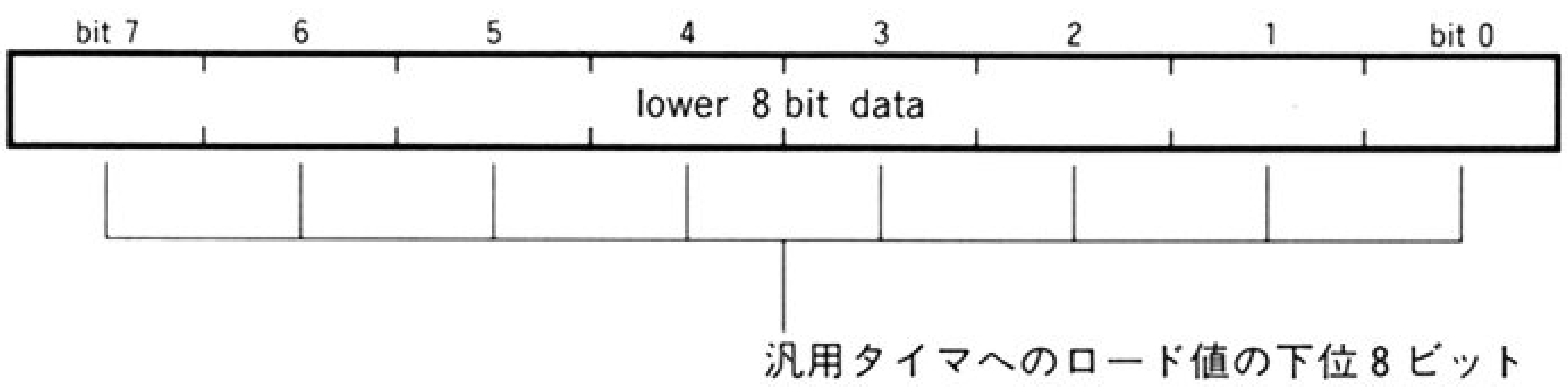
●図……31　R76 (Write) : Play-back Counter value(L)



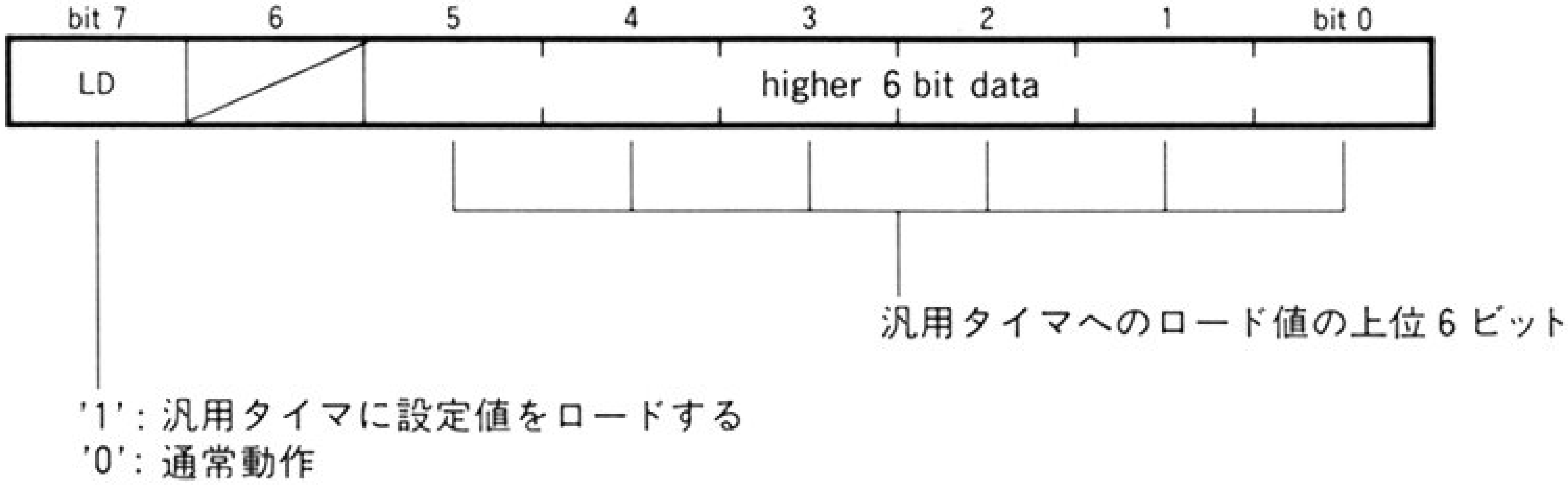
●図……32　R77 (Write) : Play-back Counter value(H)



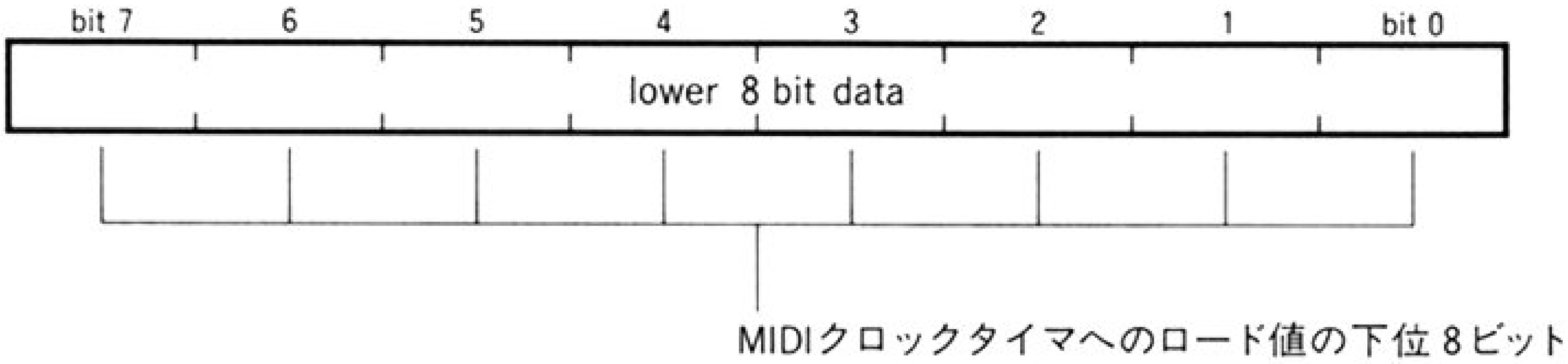
●図……33　R84 (Write) : General Timer value(L)



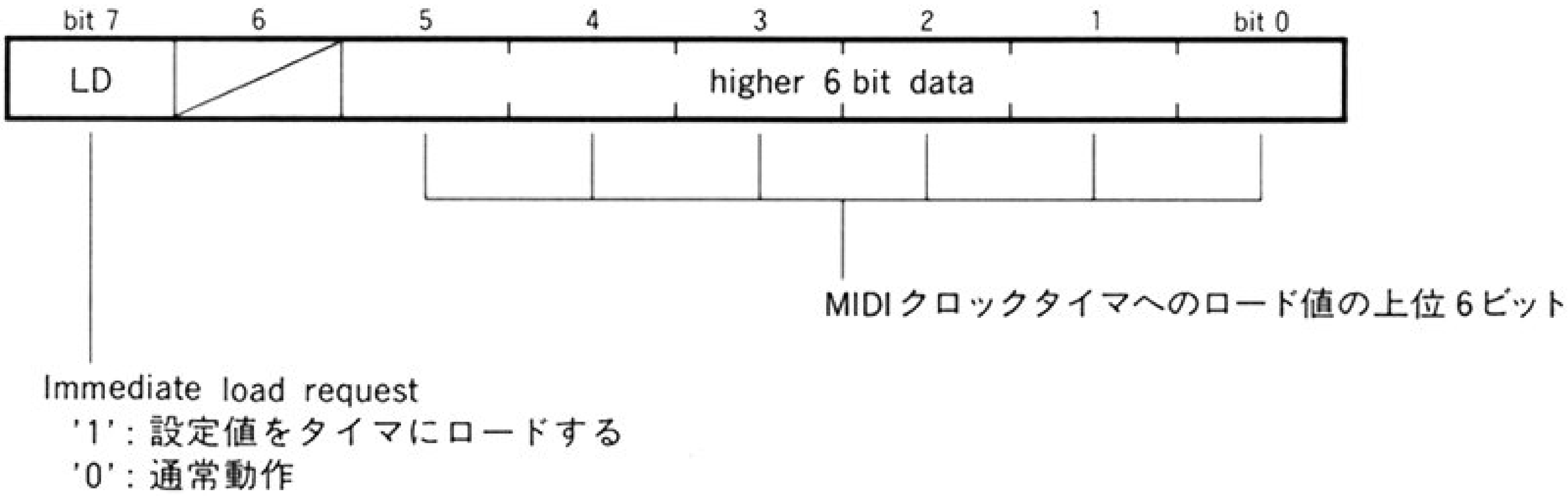
●図……34　R85 (Write) : General Timer value(H)



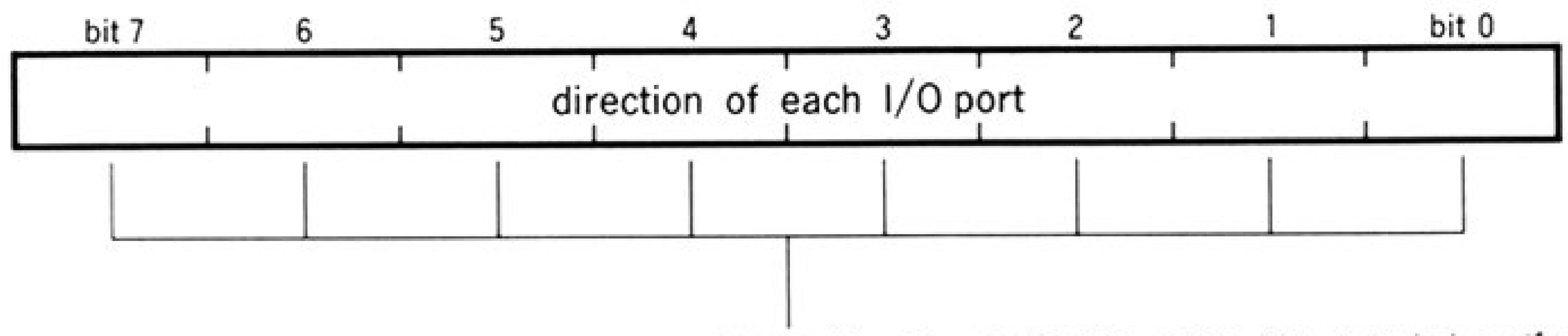
●図……35　R86 (Write) : MIDI-clock Timer value(L)



●図……36　R87 (Write) : MIDI-clock Timer value(H)

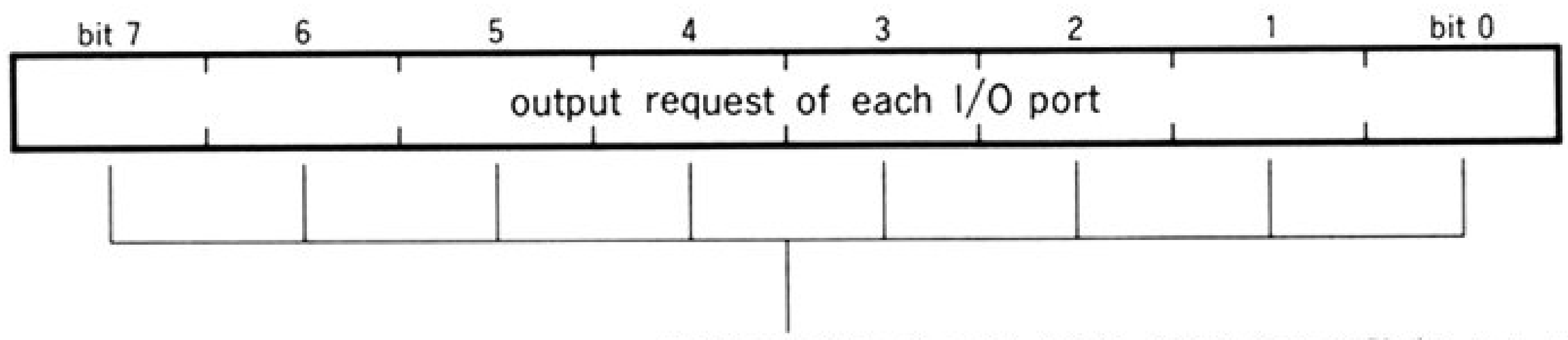


●図……37　R94 (Write) : External I/O direction



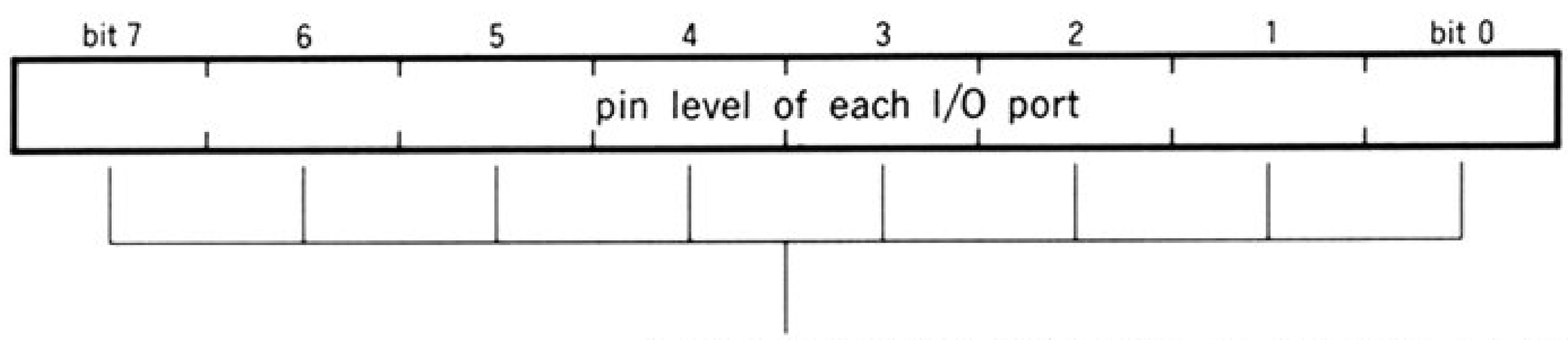
YM3802の持つ8つのI/Oピンのそれぞれを入出力いずれで使用するか決める
'1' : output
'0' : input

●図……38　R95 (Write) : External I/O output data



YM3802の持つ8つのI/Oピンのうち出力に設定したものの状態を設定する
'1' : 'H' レベル
'0' : 'L' レベル

●図……39　R96 (Write) : External I/O input data



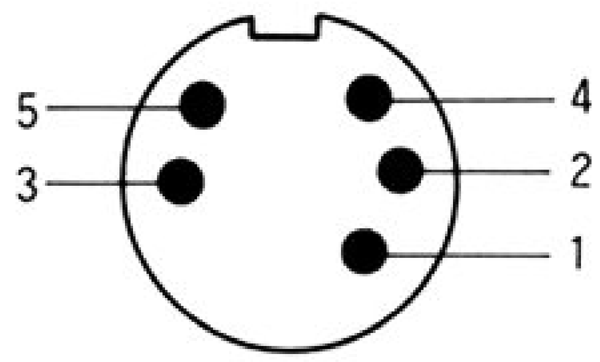
YM3802の持つ8つのI/Oピンのうち入力に設定したものの状態を読み出す
'1' : 'H' レベル
'0' : 'L' レベル

●6　コネクタ信号配置

CZ-6BM1の各コネクタの信号配置は図40のようになっています。

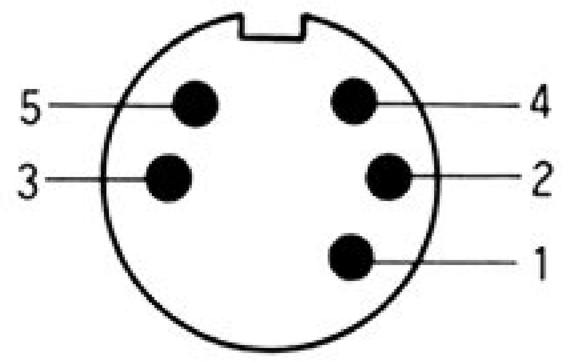
●図……40　MIDI ボードのコネクタの信号配置

• MIDI OUT, MIDI THRU



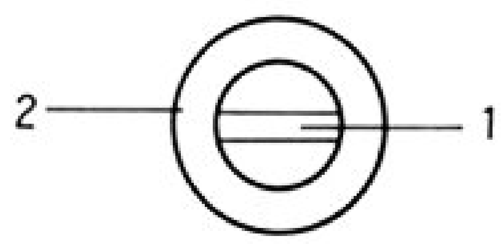
No.	信 号 名	備　考
1	N. C.	
2	GND	
3	N. C.	
4	MIDI 信号 ＋	OUT
5	MIDI 信号 －	IN

• MIDI IN



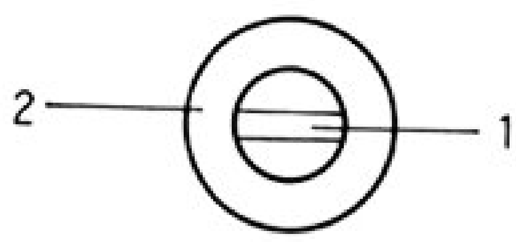
No.	信 号 名	備　考
1	N. C.	
2	GND	
3	N. C.	
4	MIDI 信号 ＋	IN
5	MIDI 信号 －	OUT

• TAPE SYNC OUT



No.	信 号 名	備　考
1	TAPE SYNC OUT	OUT
2	GND	

• TAPE SYNC IN



No.	信 号 名	備　考
1	TAPE SYNC IN	IN
2	GND	

●7　回路図

CZ-6BM1 の回路図を図 41 に示します。

●図……41　CZ-6M1 の回路図（巻末参照）

パラレルボード（CZ-6BN1）

イメージスキャナとの接続を主な目的として設計されたパラレルボードは，大量のデータの入出力を高速に行うことができるインタフェースボードです。X68000同士の接続など，さまざまな用途が期待されるボードでもあります。

パラレルボードは，シャープのカラーイメージスキャナ（CZ-8 NS 1）や日本電気製のイメージスキャナ（PC-IN 500 シリーズ）と接続し，高速のデータ転送を行うことができるボードです。スキャナと接続しないときは非絶縁型のデジタル入出力ボードとして使うこともできます。

1　仕　様

主要 LSI	
パラレル入出力 IC	μPD 8255 A
入出力バッファ IC	SN 74 LS 244
パラレルインタフェース	
入出力データライン	各 8 ビット
ハンドシェークライン	各 2 ビット
その他（ステータス等）	4 ビット
I/O コネクタ	37 ピン D-SUB コネクタ

電源	
電源電圧	＋5 V
消費電流	約 370 mA
動作温度範囲	10〜35℃
ポートアドレス	$EAFB01〜$EAFB0F/$EAFB11〜$EAFB1F （ジャンパースイッチで選択）
割り込み	レベル 2/レベル 4（ジャンパースイッチで選択）

●2 回路概要

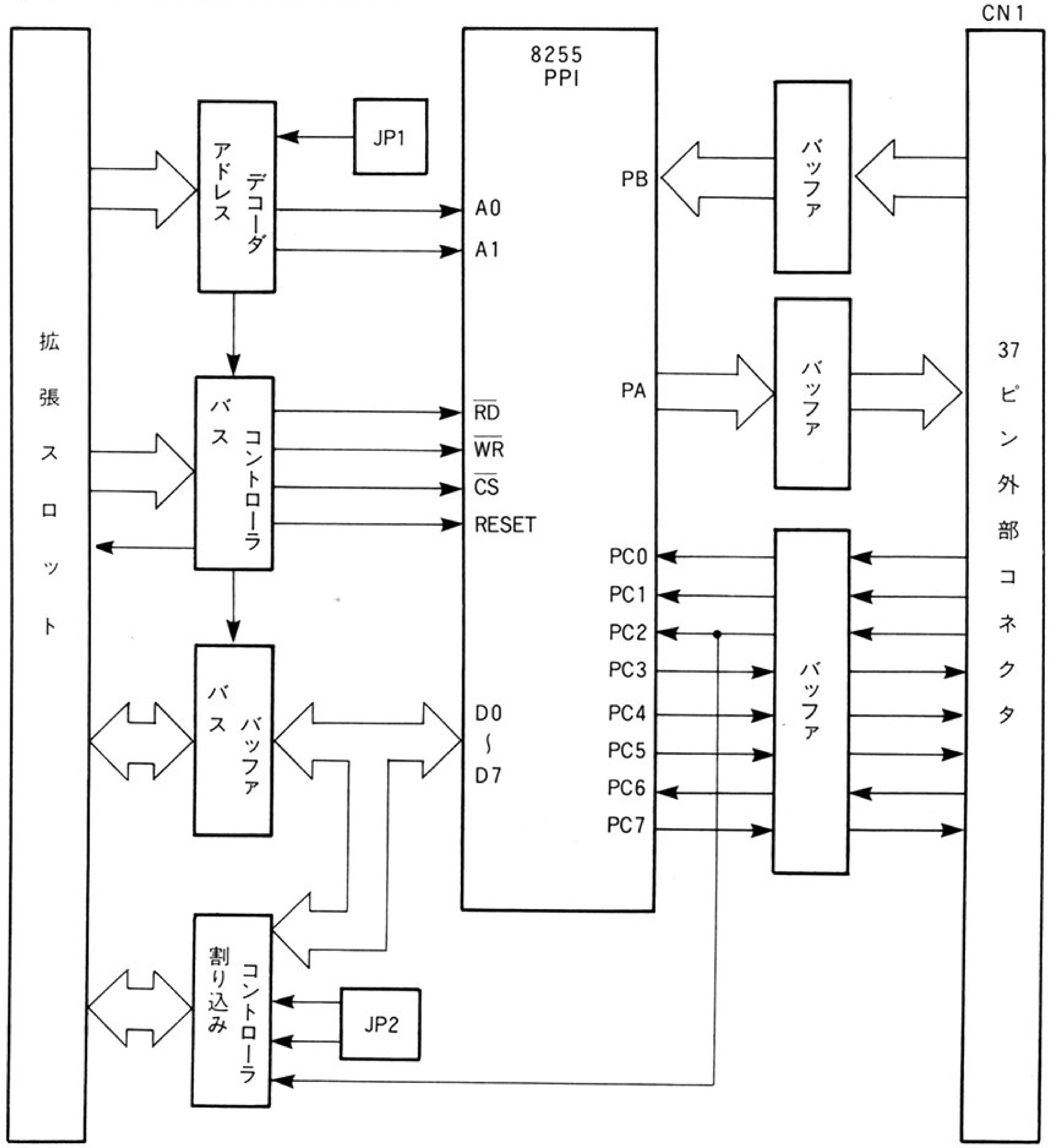
CZ-6BN1 のブロック図を図 1 に示します。パラレル入出力用の IC として，X68000 のジョイスティックインタフェースなどにも使われている μPD 8255 A が使用されており，バッファ IC (74 LS 244) を通して 37 ピンの D-SUB コネクタと接続されています。このインタフェースはイメージスキャナのパラレルインタフェース仕様に準拠しており，イメージスキャナと接続すれば RS-232 C で接続した場合よりもはるかに高速なデータ入力が可能となります。

8255 にはいくつかの動作モードがありますが，イメージスキャナとの接続ではグループ A をモード 1 の出力モードで，グループ B をモード 0 の入力モードで使うようになっています。このモードでは，8255 の I/O ピンのうち PC 3, 6, 7 の 3 つがグループ A の制御信号として使用されるようになります。

余っている PC 0, 1, 2, および PC 4, 5 の 5 本はそれぞれ入力と出力のいずれにもプログラム可能ですが，CZ-6BN1 では外付けのバッファの向きが固定であるため，PC 0, 1, 2 を入力，PC 4, 5 を出力として使用することになります。これらの端子はスキャナとの接続時にはスキャナ側のステータス入力やスキャナへのリセット出力信号などとして使われます。

これらのうち，8255 の PC 2 端子への入力信号は割り込み発生回路にも接続されています。スキャナと接続した場合，この信号はスキャナ側の IBF（入力バッファフル）信号となっており，X68000 側から送ったデータをスキャナが受け取ったときに X68000 本体に割り込みをかけることができます。

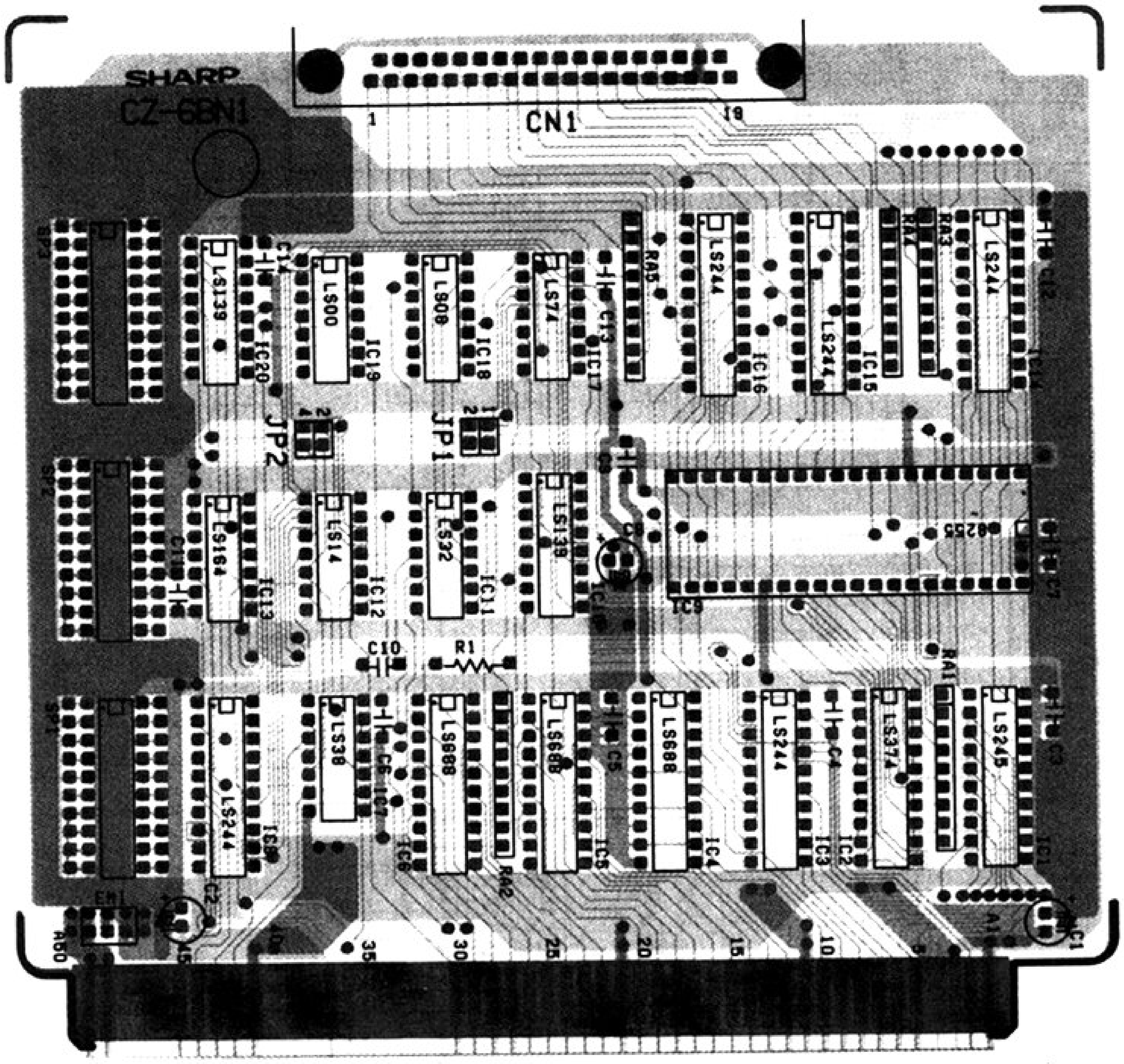
●図……1　CZ-6BN1のブロック図



●3　主要部品配置

CZ-6BN1の部品配置を図2に示します。

●図……2　CZ-6BN1の部品配置



●4　設　定

❹·1　ポートアドレスの設定

CZ-6BN1のポートアドレスはジャンパースイッチ（JP1）で切り替えることができるようになっています。JP1が1側のとき$EAFB00～$EAFB0F，2側のとき$EAFB10～$EAFB1Fのエリアが使用されます。

④·2 割り込みレベルの設定

割り込みレベルとして2と4のいずれを使うかをジャンパースイッチ(JP2)で選択することができます。2側にするとレベル2が，4側にするとレベル4が使用されるようになります。なお，割り込みマスクレジスタ（$EAFB09/$EAFB19）によって割り込みの発生をさせないようにすることもできます。このときはJP2の選択に関係なく，割り込み信号を他のボードで使用してもかまいません。

●5 I/Oマップ

CZ-6BN1のI/Oマップを図3に示します。$EAFB01/$EAFB11～$EAFB07/$EAFB17までがμPD 8255 Aのポートおよびコントロールワードレジスタ，$EAFB09/$EAFB19が割り込みマスクレジスタ，$EAFB0B/$EAFB1Bが割り込みベクタレジスタになっています。

●図……3　パラレルボードのI/Oアドレスマップ

デバイス	アドレス	レジスタ	7	6	5	4	3	2	1	0
8255	ベースアドレス +$01	ポートA	SEND DATA							
	+$03	ポートB	RECEIVE DATA							
	+$05	ポートC	OBF	ACK	CACK	RESET	INTR	IBF	READY	DCN
	+$07	コントロールワードレジスタ								
	+$09	割り込みマスクレジスタ								INT
	+$0B	割り込みベクタ番号レジスタ	VECT							

ベースアドレス：$EAFB00（1枚目）/$EAFB10（2枚目）

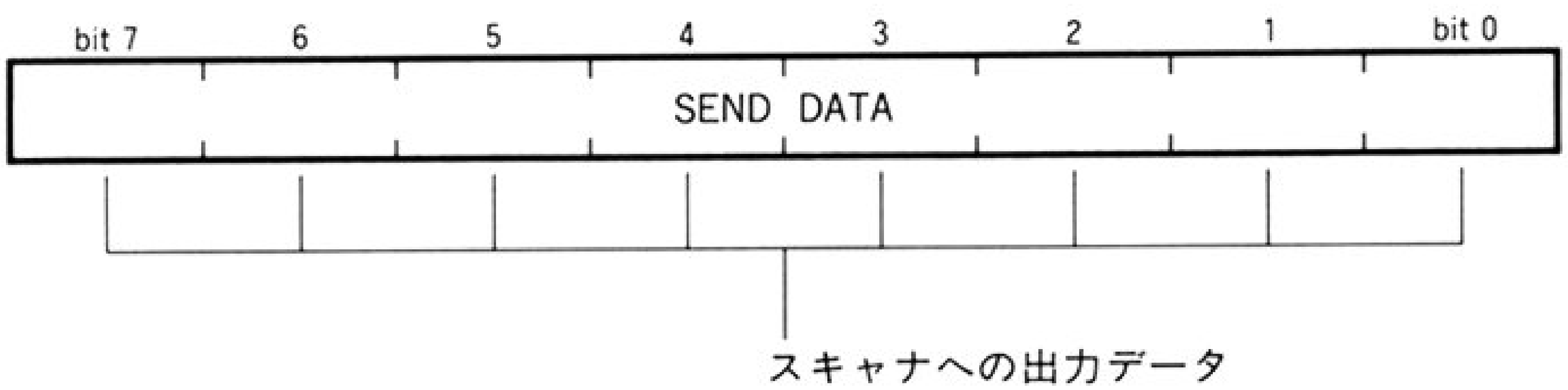
❺·1　8255のレジスタ

8255のレジスタのビット配置を図4～7に示します。スキャナとの接続では，グループAをモード1で，グループBはモード0，ポートCの下位ビットは入力としてプログラムするようにします。

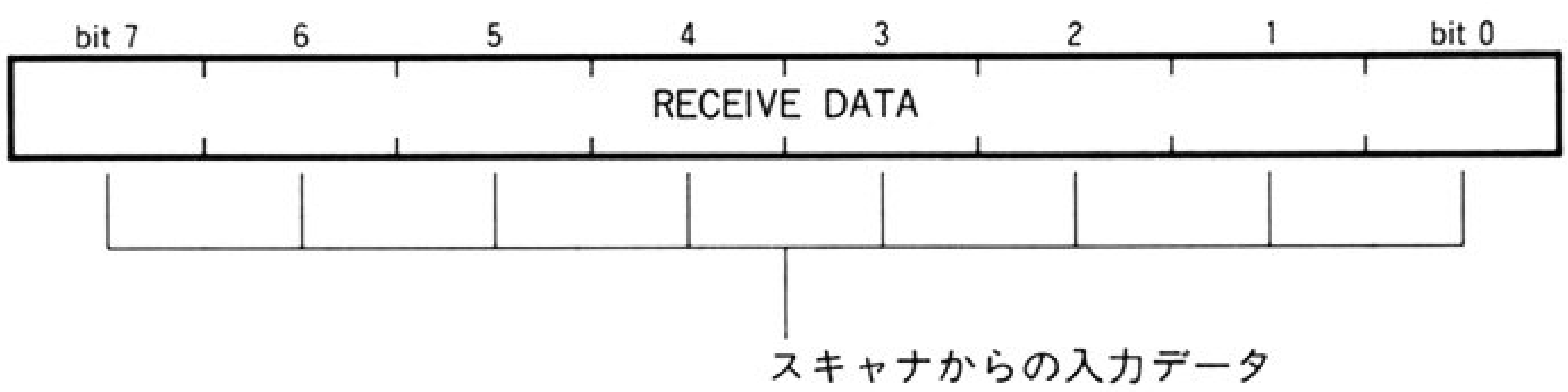
パラレルボードでは，ポートCが入出力入り乱れているため，グループAをモード1の出力モード以外にプログラムしないようにするか，あるいはポートCをすべてモード0の入力ポートとしてプログラムしないとポートCと入力バッファ（LS 244）の出力同士の衝突が起きてしまいます。

8255の各モードでの動作やプログラミング方法については拙著『Inside X68000』のジョイスティックインタフェースの章でも触れていますので参考にしてください。

●図……4　8255 ポート A



●図……5　8255 ポート B



●図……6　8255 ポート C

bit 7	6	5	4	3	2	1	bit 0
OBF	ACK	CACK	RESET	INTR	IBF	READY	DCN

DCN (bit 0): スキャナが接続されているか否かを示す
'1'：接続されていない
'0'：接続されている

READY (bit 1): スキャナのレディ信号の状態を示す
'1'：Busy 状態
'0'：Ready 状態

IBF (bit 2): スキャナのIBF（Input Buffer Full）信号の状態を示す
'1'：バッファは空
'0'：バッファにデータが入っている

INTR (bit 3): ポートAを使用したデータの受け渡し状態を示す
'1'：X68000がポートAにデータをセットし始めた
'0'：スキャナがデータを引き取り終えた
この信号はスキャナは使用していない

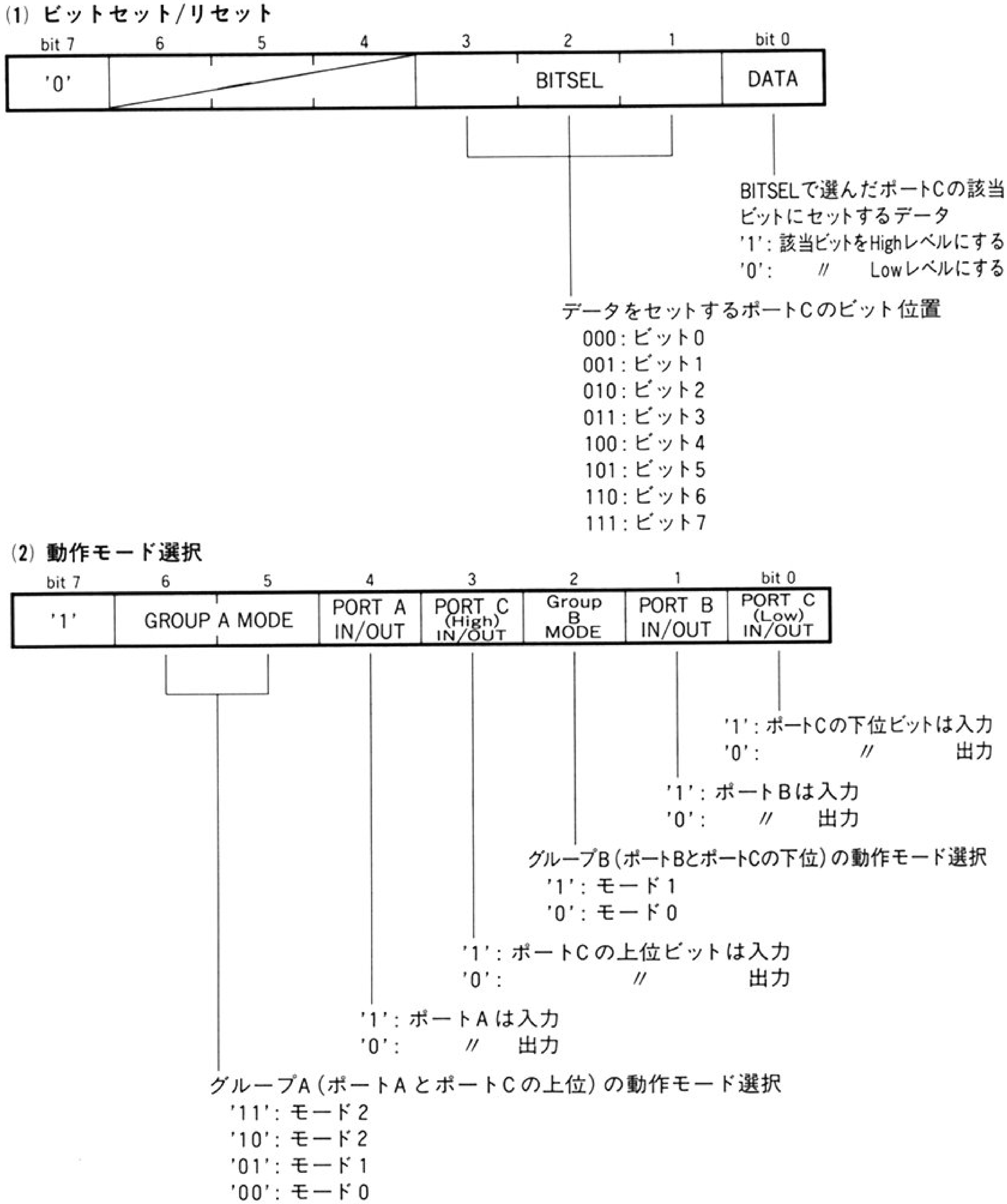
RESET (bit 4): スキャナリセット
'1'：通常動作
'0'：スキャナをリセットする

CACK (bit 5): スキャナからの送信に対する X68000 からの応答出力
'1'：通常
'0'：スキャナからのデータを受け取った

ACK (bit 6): スキャナからの応答信号入力
'1'：通常
'0'：X68000 から送ったデータを受け取った

OBF (bit 7): ポートA（出力バッファ）の状態
'1'：出力バッファは空
'0'：出力バッファにデータが入っている

●図……7　8255 コントロールワードレジスタ

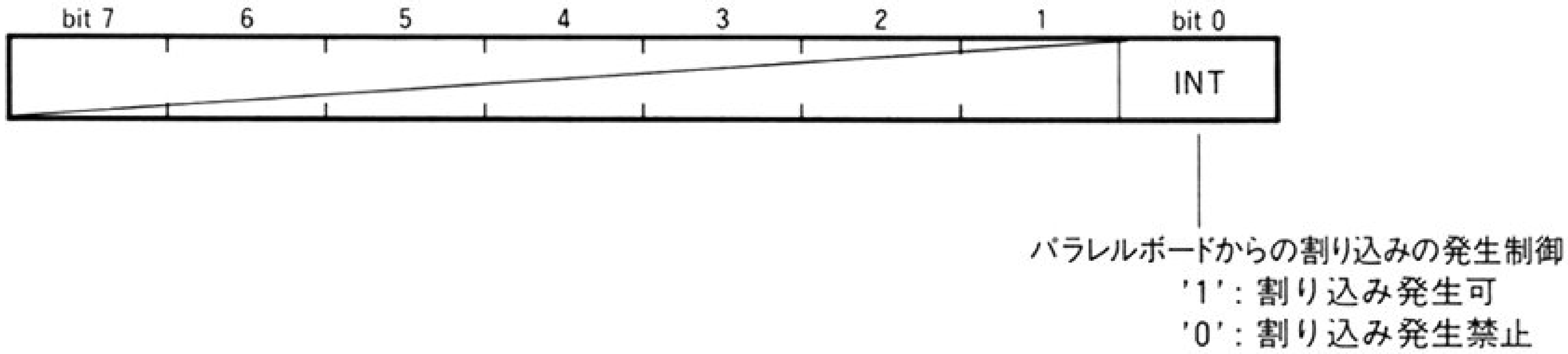


❺·2　割り込みマスクレジスタ

割り込みマスクレジスタのビット配置を図8に示します。このレジスタは最下位ビットのみが有効で，このビットを'1'にすると割り込み発生が許可，'0'にすると禁止になります。リセッ

ト後，このビットは'0'になっています。

●図……8　割り込みマスクレジスタ

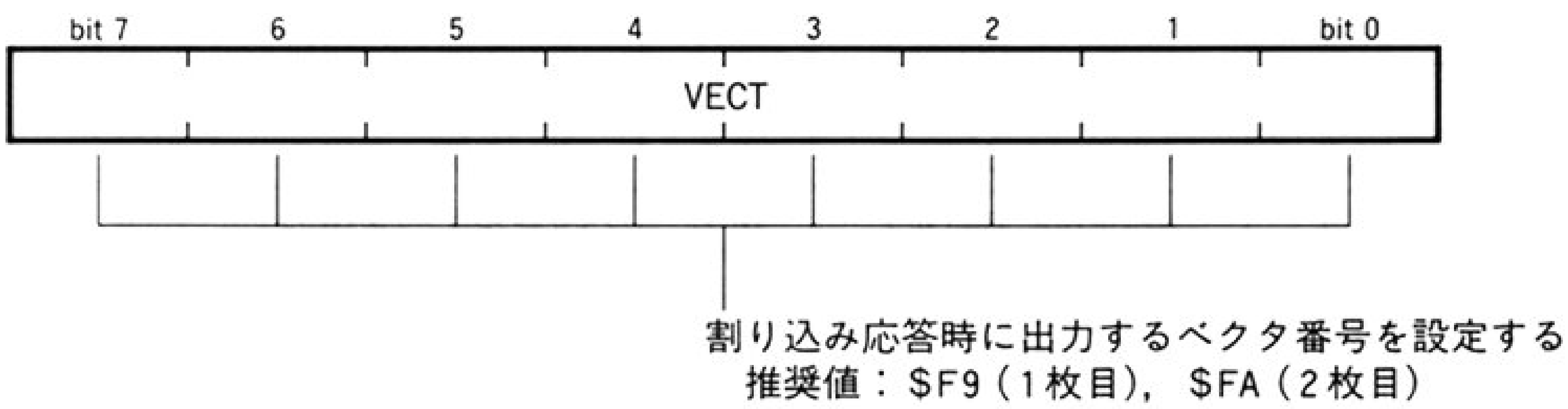


❺・3　割り込みベクタ番号レジスタ

割り込み応答サイクルで発生するベクタ番号を設定します(図9)。ベクタ番号はハード的にはX68000内部のデバイスやソフトウェアのTRAP命令などで使用しないものであればなんでもかまいません。X68000では$6C～$FFが未使用ベクタになっていますが，CZ-6BN1のサポートソフトウェアではベクタ番号として$F9（1枚目），$FA（2枚目）のうちいずれかを使用するものとしています。

ベクタ番号の設定は，必ず割り込みマスクレジスタで割り込みを許可にする前に行ってください。

●図……9　割り込みベクタレジスタ



●6　コネクタ信号配置

CZ-6BN1の37ピンD-SUBコネクタのピン配置を図10に示します。

●図……10　パラレルボードのコネクタ信号配置

端子番号	ポート番号	信　号　名	入/出力	機　　　能
1	PB 0	SEND DATA 0	入力	パラレルデータ入力
2	PB 1	SEND DATA 1	入力	パラレルデータ入力
3	PB 2	SEND DATA 2	入力	パラレルデータ入力
4	PB 3	SEND DATA 3	入力	パラレルデータ入力
5	PB 4	SEND DATA 4	入力	パラレルデータ入力
6	PB 5	SEND DATA 5	入力	パラレルデータ入力
7	PB 6	SEND DATA 6	入力	パラレルデータ入力
8	PB 7	SEND DATA 7	入力	パラレルデータ入力
9	NC			未使用
10	GND	GND		信号グランド
11	GND	GND		信号グランド
12	GND	GND		信号グランド
13	GND	GND		信号グランド
14	GND	GND		信号グランド
15	GND	GND		信号グランド
16	GND	GND		信号グランド
17	GND	GND		信号グランド
18	GND	GND		信号グランド
19	PA 0	RECEIVE DATA 0	出力	パラレルデータ出力
20	PA 1	RECEIVE DATA 1	出力	パラレルデータ出力
21	PA 2	RECEIVE DATA 2	出力	パラレルデータ出力
22	PA 3	RECEIVE DATA 3	出力	パラレルデータ出力
23	PA 4	RECEIVE DATA 4	出力	パラレルデータ出力
24	PA 5	RECEIVE DATA 5	出力	パラレルデータ出力
25	PA 6	RECEIVE DATA 6	出力	パラレルデータ出力
26	PA 7	RECEIVE DATA 7	出力	パラレルデータ出力
27	PC 4	RESET	出力	外部リセット出力
28	PC 5	CACK	出力	Aポート出力　アクノリッジ
29	PC 6	ACK	入力	Bポート入力　アクノリッジ
30	PC 7	OBF	出力	Aポート出力バッファ　フル
31	PC 0	DCN	入力	外部機器の接続確認入力信号
32	PC 1	READY	入力	外部機器の動作確認入力信号
33	PC 2	IBF	入力	Bポート入力バッファ　フル
34	PC 3	INTR	出力	Aポート出力バッファ　フル
35	NC			未使用
36	GND	GND		信号グランド
37	NC			未使用

プラグ：DCLC-J37PAF-10L8（日本航空電子）相当品

●7 パラレルケーブル接続図

ボードに付属するパラレルケーブル（スキャナとの接続ケーブル）の接続図を図11に示します。

●図……11　パラレルケーブルの接続（ケーブル例 1FVV-SB・18P）

57E-30360	線色	マーク	17JE-23370-02
1	橙	アカA　A1	1
2		アオ　//	2
3	灰	アカ　//	3
4		アオ　//	4
5	白	//	5
6		//	6
7	黄	//	7
8		//	8
9	桃	//	9
10		//	10
11	橙	A2	11
12		//	12
13	灰	//	13
14		//	14
15	白	//	15
16		//	16
17	黄	//	17
18		//	18
19	桃	//	19
20		//	20
21	橙	A3	21
22		//	22
23	灰	//	23
24		//	24
25	白	//	25
26		//	26
27	黄	//	27
28		//	28
29	桃	//	29
30		//	30
31	橙	A4	31
32		//	32

33	灰	アカ	〃	33
34		アオ	〃	34
35	白	アカ	〃	35
36		アオ	〃	36
37			〃	37
38				

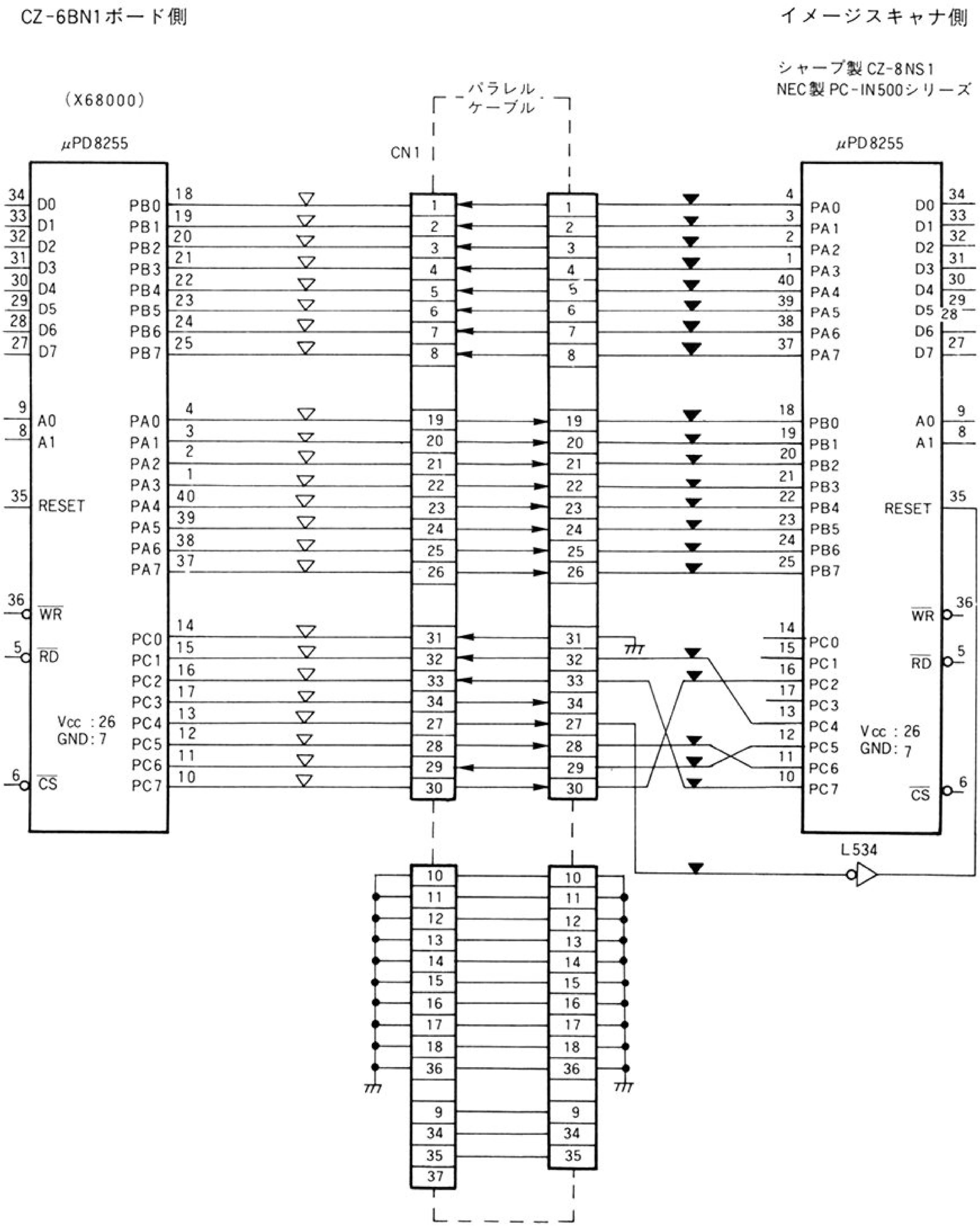
●8 イメージスキャナ接続例

イメージスキャナと接続したときの，インタフェース部分の回路を図12に示します。当然のことながら，イメージスキャナ側のコネクタより内側の部分はモデルチェンジなどによって変更されることがありますので，この図はあくまでも参考程度と考えてください。

●9 回路図

CZ-6BN1の回路図は図13のようになっています。

●図……12　イメージスキャナ接続時のインタフェース回路



●図……13　CZ-6BN1の回路図（巻末参照）

ビデオボード (CZ-6BV1)

ビデオボードはX68000の15KHzモードの画面をビデオデッキやTVなどで表示できるようにするアダプタボードです。グラフィックを大画面に映したり，ビデオに録画するなどの目的には最適なボードです。

ビデオボードは, X68000のアナログRGB信号をコンポジットビデオ信号やS映像信号(Y-C分離信号)に変換するアダプタです。X68000の15KHzモード時の画面をビデオデッキなどで録画したり，大型テレビや液晶ビジョンなどに接続することができるようになります。

1 仕様

主要LSI	
NTSCエンコーダIC	CXA1145P
同期信号発生IC	CXD1217M
入力信号	アナログRGB信号
	TVコントロール信号
出力信号	アナログRGB信号
	TVコントロール信号
	S映像信号
	コンポジットビデオ信号

I/O コネクタ	
アナログ RGB	15 ピン D-SUB
TV コントロール	8 ピン DIN
S 映像信号	4 ピンミニ DIN
コンポジットビデオ	ピンジャック
電源	
電源電圧	＋5 V/＋12 V
消費電流	
＋5 V	100 mA
＋12 V	150 mA
動作温度範囲	0～35℃

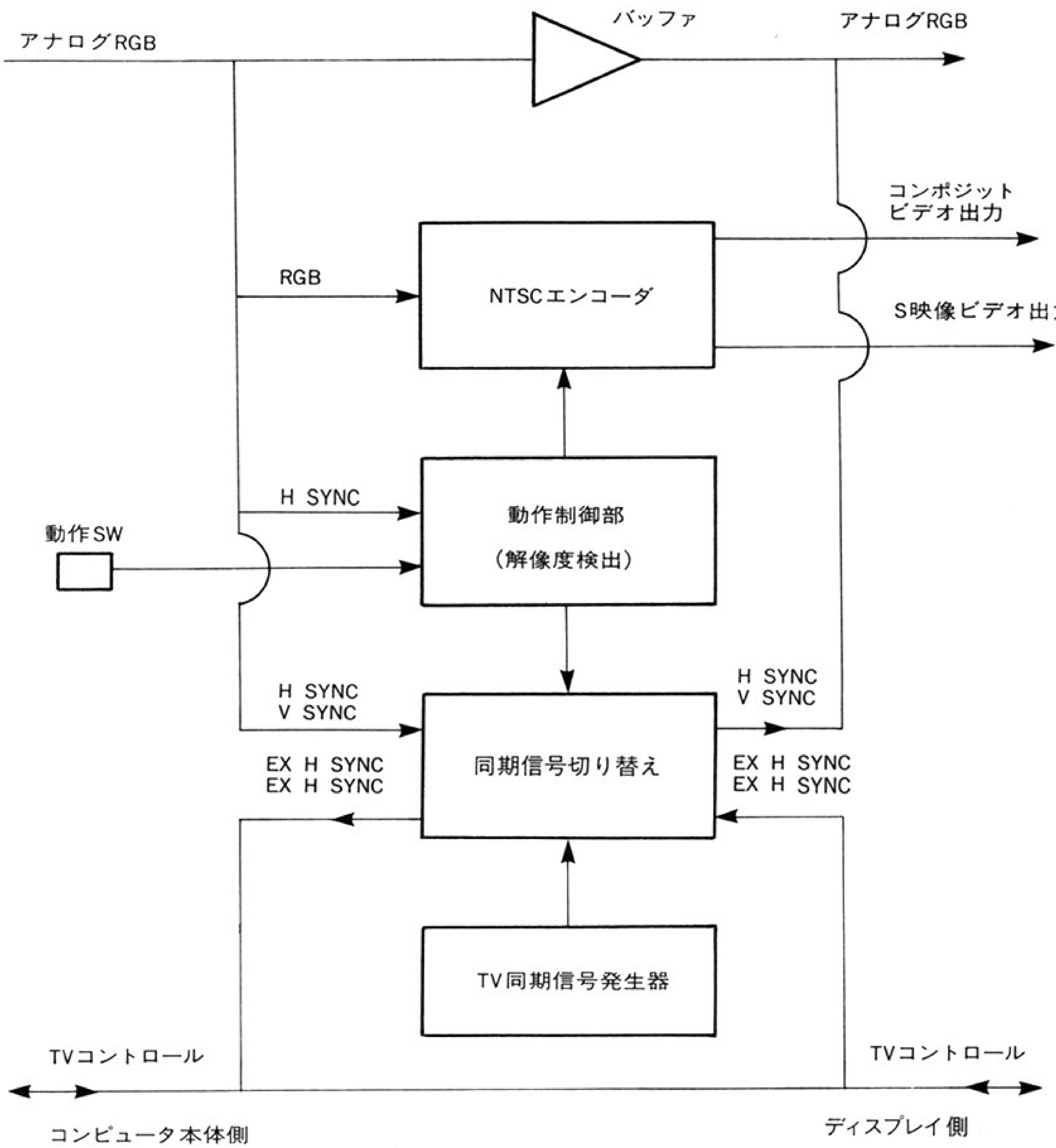
●2 回路概要

CZ-6BV1のブロック図を図1に示します。X68000本体から出力されるアナログRGB信号はそのままバッファされてアナログRGB信号として出力されるほか，NTSCエンコーダに入力されます。ビデオボードはX68000の拡張スロットに挿入されますが，拡張バスからは単に電源を受け取っているだけで，I/Oポートなどはまったく設けられておりません。

X68000の画面が15 KHzモードになっていることを検出すると，NTSCエンコーダが動作し，コンポジットビデオ信号やS映像信号を出力するようになります。ボードにはビデオボード動作スイッチが取り付けられており，ビデオボードの動作を強制的にOFFすることもできるようになっています。

X68000の場合，TV映像とアナログRGB信号の合成は専用ディスプレイ内で行われているため，ビデオボードを経由して接続したS映像信号やコンポジット映像信号用のディスプレイ（ビデオディスプレイと呼ぶことにします）上にはTV画像は表示されません。したがって，コンピュータモード，スーパーインポーズモード，テレビモードのいずれにしたときでもビデオディスプレイ上にはX68000の画面（コンピュータ画面）が表示され，スーパーインポーズ表示やテレビ画面の表示は行われません。

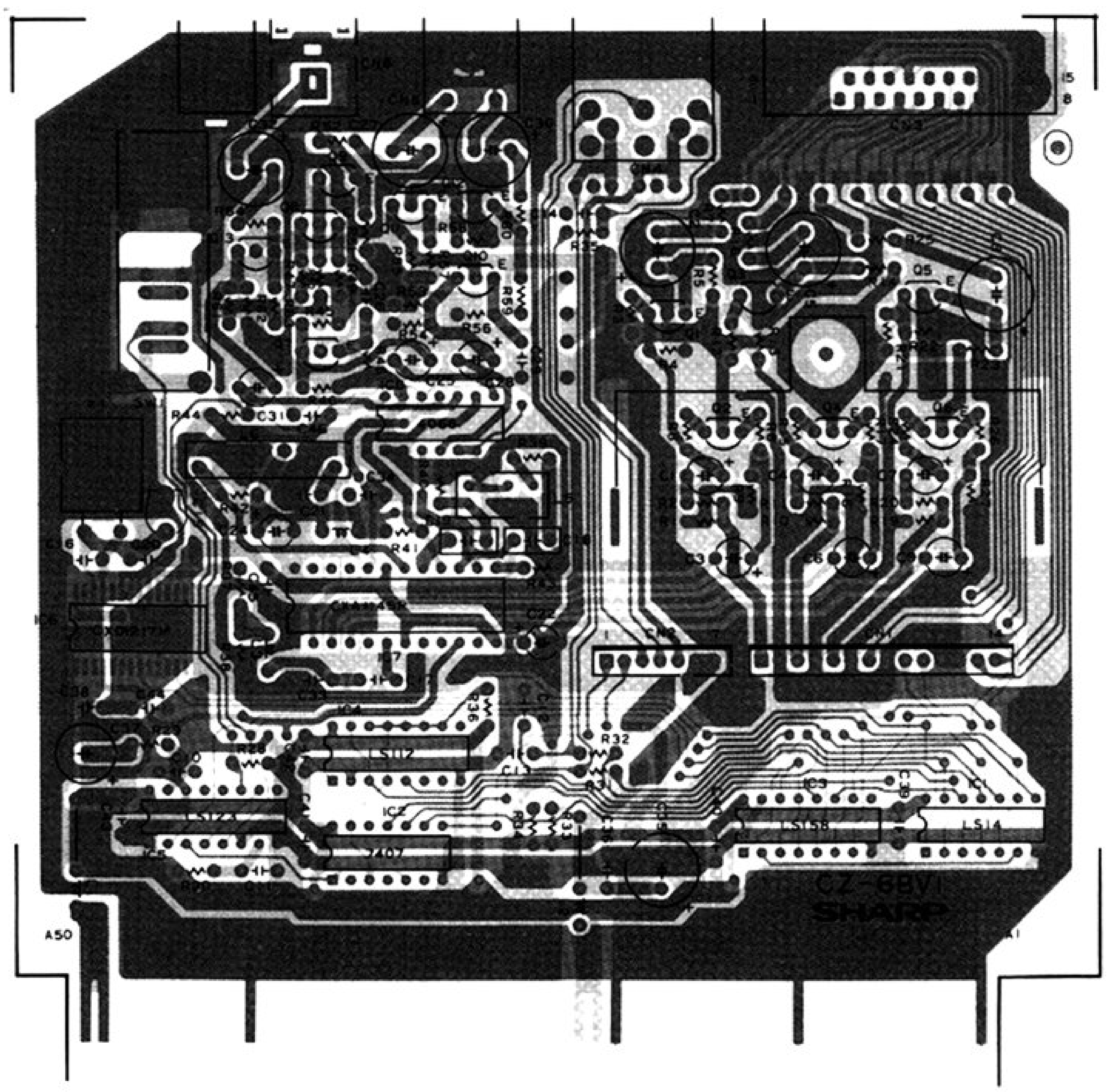
●図……1　CZ-6BV1 のブロック図



3 主要部品配置

CZ-6BV1 の部品配置を図 2 に示します。

●図……2　CZ-6BV1の部品配置



●4 設　定

❹・1 ビデオボード動作

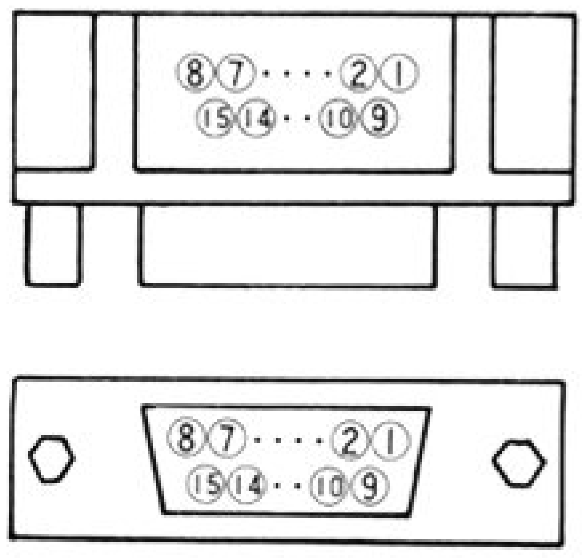
ビデオボード動作スイッチによってビデオボードの動作ON/OFFを行うことができます。OFFにした場合，ビデオディスプレイへの表示は停止されます。

●5 コネクタ信号配置

CZ-6BV1のコネクタ信号配置を図3に示します。

●図……3　ビデオボードのコネクタ信号配置

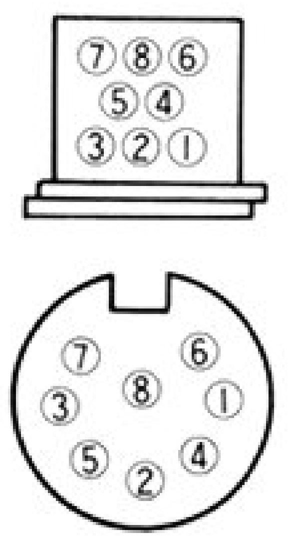
・アナログRGB信号コネクタ，ケーブル



端子No.	信号名	方向	備考
1	R OUT	C→M	アナログ0.7Vp-p(75Ω終端時)
2	GND	−	グランド
3	R OUT	C→M	アナログ0.7Vp-p(75Ω終端時)
4	GND	−	グランド
5	R OUT	C→M	アナログ0.7Vp-p(75Ω終端時)
6	GND	−	グランド
7	YS	C→M	コンピュータデータの有無を示す
8	GND	−	グランド
9	N.C	−	非接続
10	AUDIO L	C→M	音声信号　左
11	AUDIO R	C→M	音声信号　右
12	GND	−	グランド
13	N.C	−	非接続
14	$\overline{\text{HSYNC}}$	C→M	水平同期信号TTLレベル
15	$\overline{\text{VSYNC}}$	C→M	垂直同期信号TTLレベル

注）方向のC→Mは，コンピュータ本体→ビデオボード→ディスプレイに信号が通ります。

・TVコントロールコネクタ，ケーブル

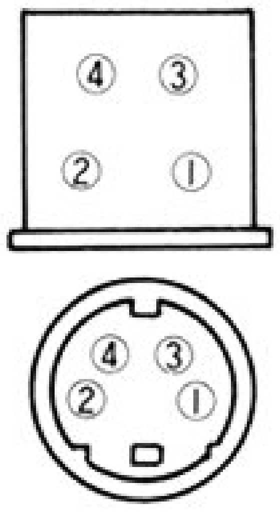


端子No.	信号名	方向	備考
1	EXHSYNC	M→C	外部水平同期信号TTLレベル
2	EXVSYNC	M→C	外部垂直同期信号TTLレベル
3	TVPOWERON/OFF	M→C	TVパワーオン/オフ信号
4	TVREMOTE	C→M	TVリモート信号
5	Vcc1	−	5V
6	GND	−	グランド
7	GND	−	グランド
8	N.C	−	非接続

注）方向のC→Mは，コンピュータ本体→ビデオボード→ディスプレイに信号が通ります。
　　方向のM→Cは，ディスプレイ→ビデオボード→コンピュータ本体に信号が通ります。

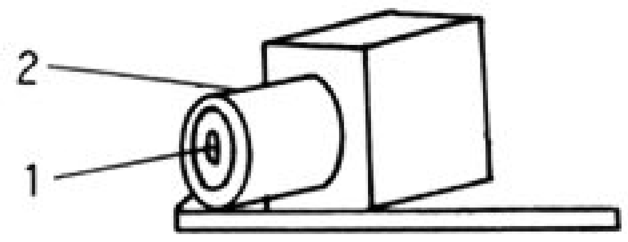
図……3　ビデオボードのコネクタ信号配置（つづき）

• S映像ビデオ出力



端子No.	信号名	I/O	備考
1	GND	―	グランド
2	GND	―	グランド
3	Y	OUT	輝度信号，1Vp-p
4	C	OUT	クロマ信号，バースト：0.286Vp-p

• コンポジットビデオ出力



端子No.	信号名	I/O	備考
1	COMP. VIDEO	OUT	コンポジットビデオ信号
2	GND	―	グランド

●6 回路図

CZ-6BV1の回路図を図4に示します。

●図……4　CZ-6BV1の回路図（巻末参照）

SCSIボード (CZ-6BS1)

SCSIは，CD-ROM，光磁気ディスク，DATなど，次世代の外部メディアへのインタフェースとして注目されています。SCSIボードはSCSIインタフェースを持たないSUPER以前の機種にSCSIをサポートするボードです。

SCSIボード（CZ-6BS1）はSCSIインタフェースを内蔵していないX68000シリーズでSCSIを利用できるようにするものです。

X68000ではSUPER，XVI以降の機種ではハードディスクインタフェースにSCSIが，それ以前の機種ではSASIが採用されていました。SCSIはSASIを拡張してANSIで標準化を行ったもので，ハードディスクだけでなくCD-ROM，光磁気ディスクなどさまざまなデバイスが標準インタフェースとしてSCSIを利用するようになっています。

1 仕様

主要LSI	
SCSIコントローラ（SPC）	MB 89352
クロック周波数	5 MHz
SCSIバス	
準拠規格	ANSI X 3.131-1986

伝送方法	不平衡型
データ転送モード	非同期転送
電源	
電源電圧	+5 V
動作温度範囲	10～35℃
ポートアドレス	$EA 0000～$EA 1 FFF
	(SCSI コントローラ：$EA0000～$EA001F)
	(SCSI ROM　　　：$EA0020～$EA1FFF)
割り込み	レベル 2/レベル 4（ジャンパースイッチで選択）
DMA	DMA チャンネル# 1

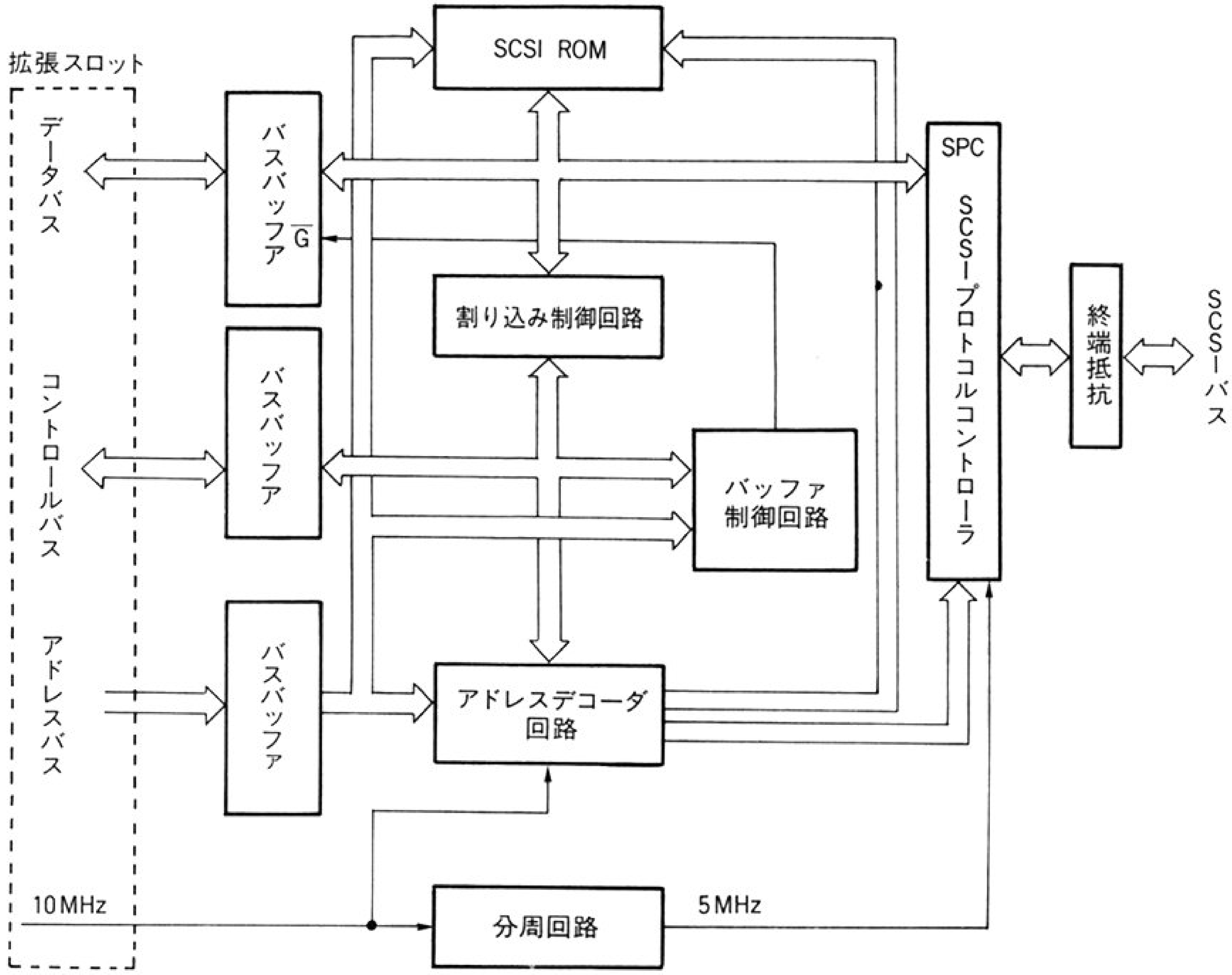
2 回路概要

CZ-6BS1 のブロック図を図 1 に示します。SCSI コントロール LSI として富士通の MB 89352 (SCSI プロトコルコントローラ：SPC と略します) が使用されています。SPC のクロックにはバス上の 10 MHz を分周して得た 5 MHz が供給されています。

SCSI ROM は SCSI バスからのブート (起動) などをサポートするためのソフトウェアが入った ROM です。SUPER 以前の機種の IPL-ROM では SCSI からのブートは考慮されていませんので，CZ-6BS1 側にプログラムの入った ROM を用意することで解決しているわけです。

CZ-6BS1 では割り込み信号と DMA のチャンネル#1 を使用します。DMA のチャンネル# 1 は SASI と共用であるため，直接 SPC を操作して SCSI バスをアクセスするときは SASI バスをアクセスするプログラムとの競合に注意する必要があります。

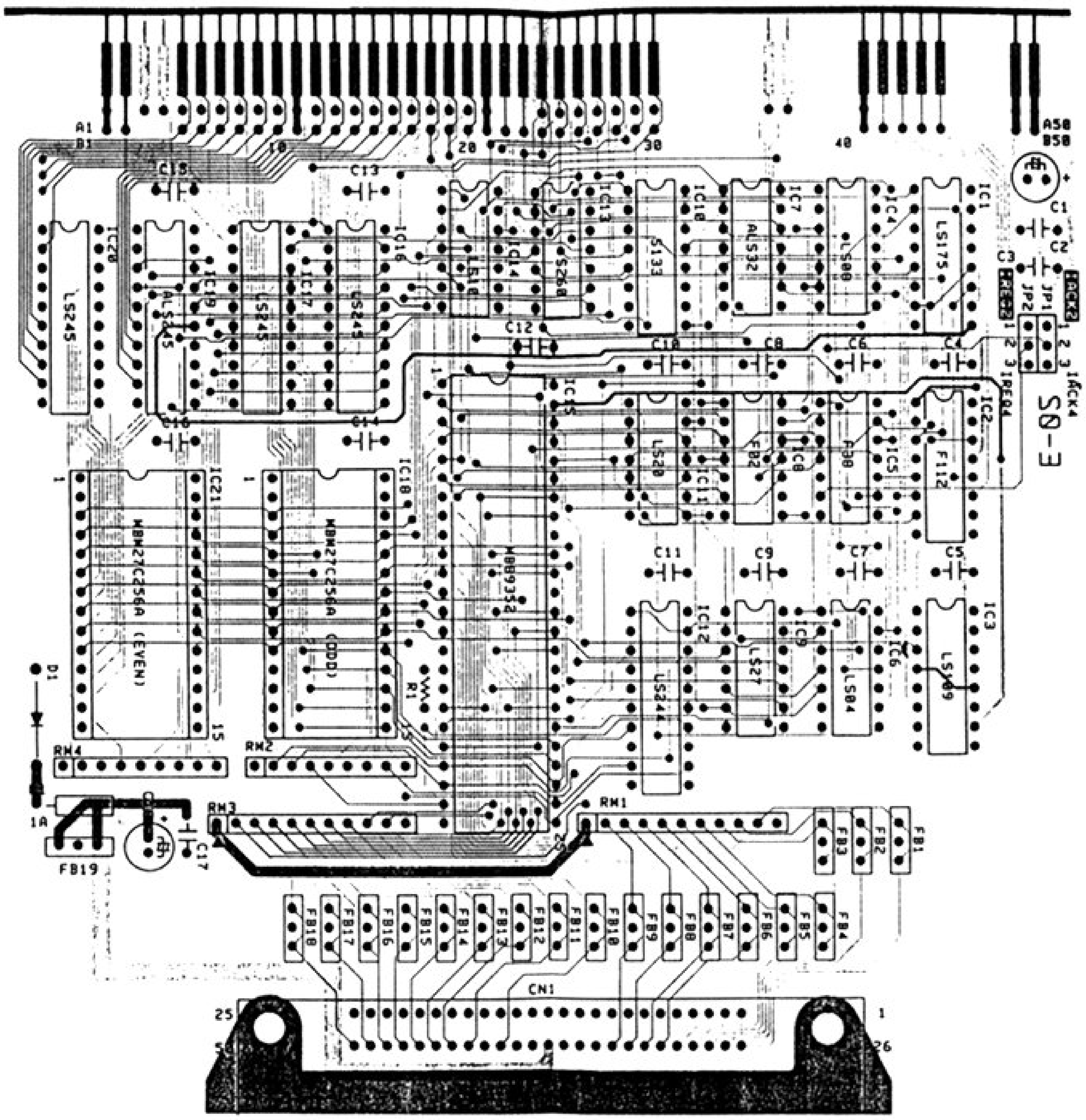
●図……1　CZ-6BS1 のブロック図



●3 主要部品配置

CZ-6BS1 の部品配置を図 2 に示します。

●図……2　CZ-6BS1 の部品配置

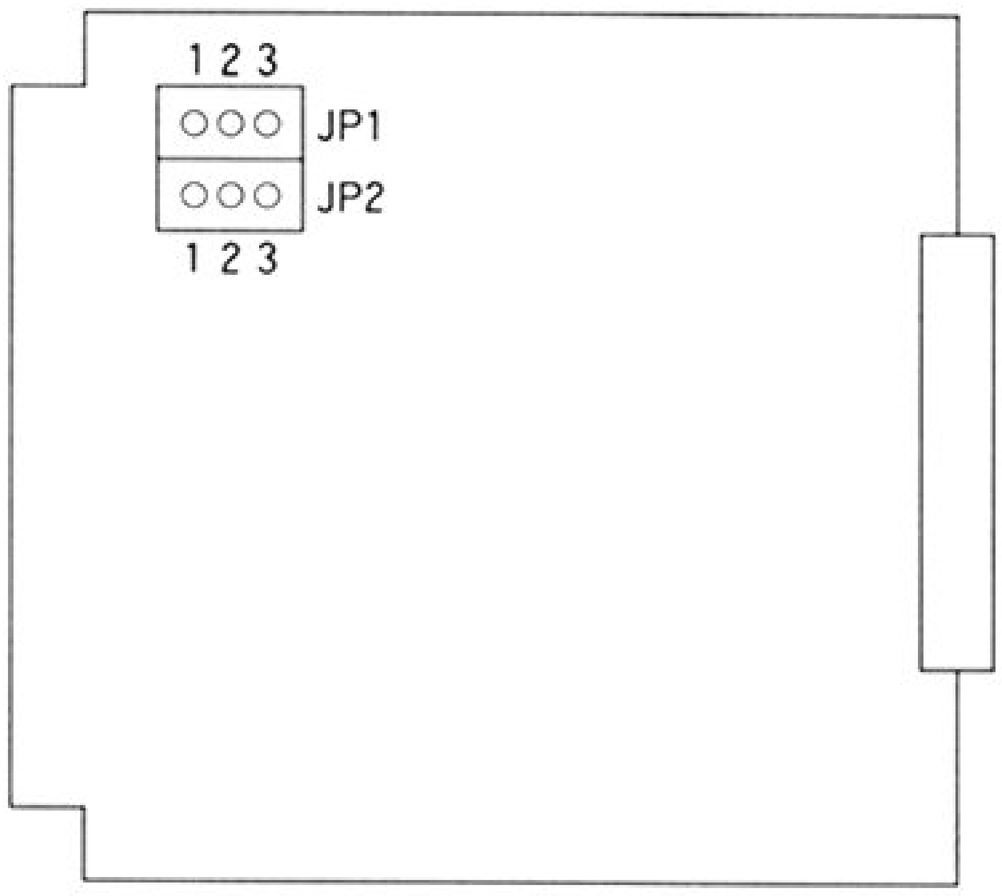


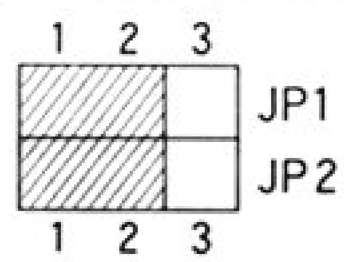
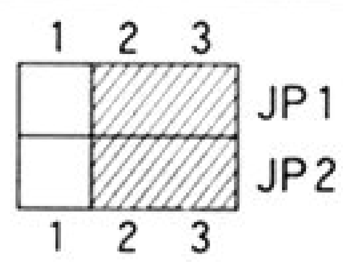
●4　設　定

❹·1　割り込みレベルの設定

CZ-6S1 は，ボード上のジャンパースイッチ（JP 1，および JP 2）によって使用する割り込みを切り替えられるようになっています（図 3）。JP1 は本体からの割り込み応答（IACK）信号を，JP2 は割り込み要求（IREQ）の切り替えを行います。

●図……3　ジャンパースイッチ（JP1，JP2）



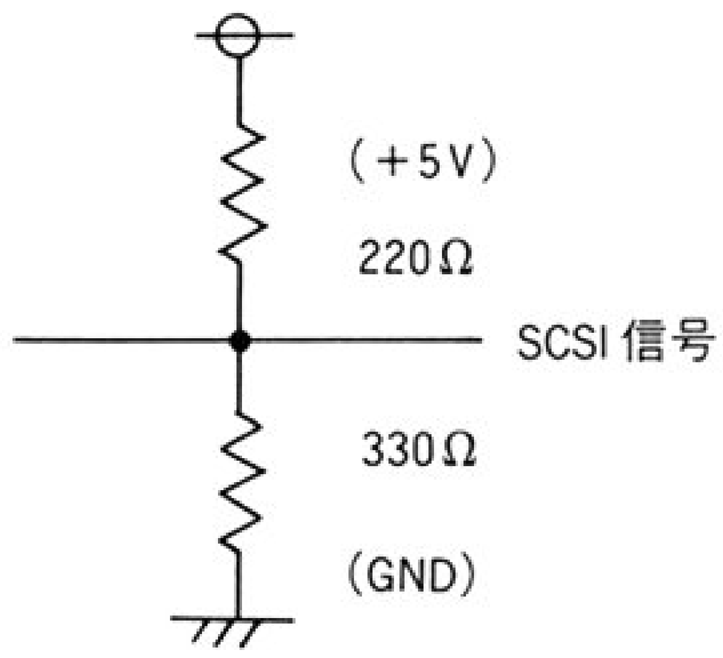
使用する割り込みレベル	設　　定
レ ベ ル　2	
レ ベ ル　4	

ジャンパースイッチの 1-2 間をショートすると割り込みレベル 2 が，2-3 間をショートすると割り込みレベル 4 が使用されるようになります。IREQ と IACK は常にペアで使用されますので，かならず同一のレベルを使用するように設定しなくてはなりません。

❹·2　終端抵抗

SCSI バスは，バス上に最大 8 台までのデバイスを接続することができるようになっています。バスは途中でT字型などに分岐することは許されておらず，両端には各信号線ごとに図 4 のような 220 Ω と 330 Ω の抵抗による処理が必要です(終端抵抗と呼びます)。CZ-6BS1 では，RM1 と RM3 が終端抵抗となっています。

●図……4　終端抵抗



ごく普通の使い方では，CZ-6BS1は端に置かれることになりますので，終端抵抗は取り付けられた状態で出荷されていますが，配線の都合などでCZ-6BS1がバスの途中にくる場合にはこの終端抵抗をはずさなくてはなりません。終端抵抗はソケットを利用して取り付けてありますので容易に着脱することができます。

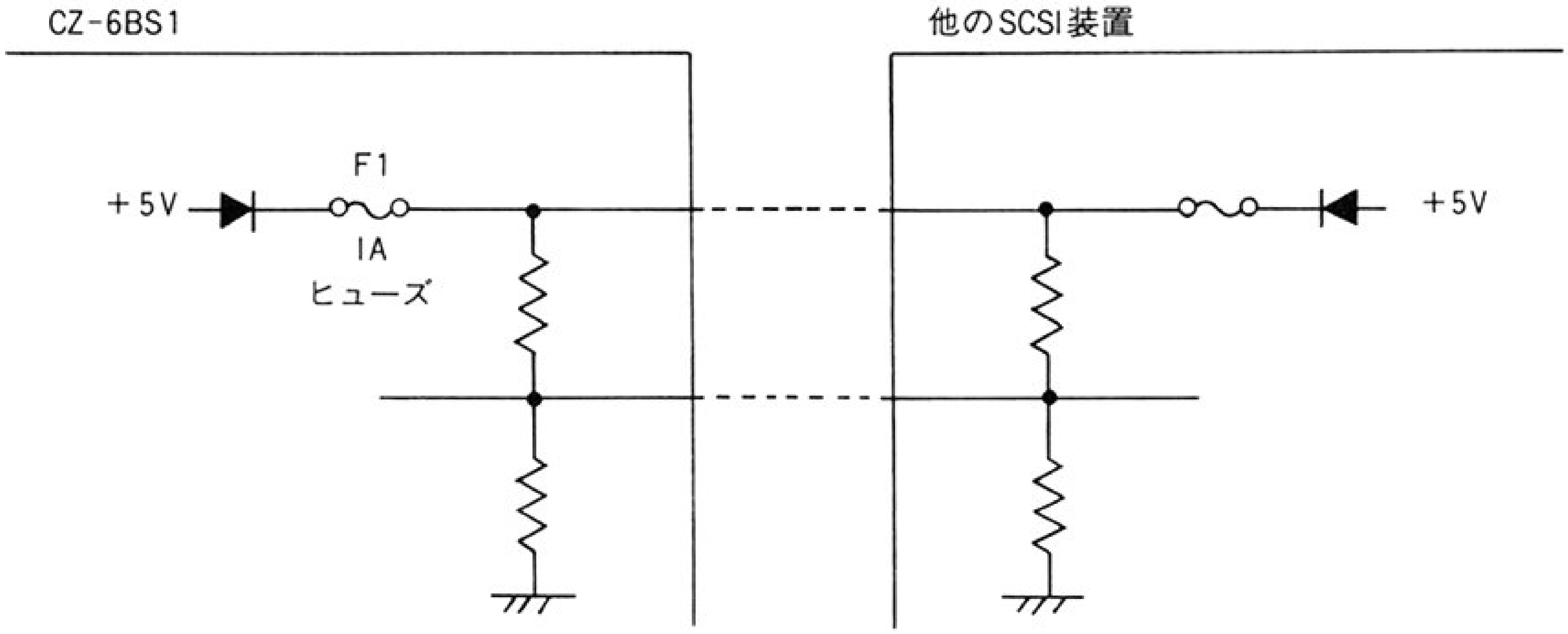
❹・3　終端抵抗用電源ライン

設定項目ではありませんが，終端抵抗がらみで少し注意が必要ですので，ここで終端抵抗用の電源（TERMPWR）について説明しておきます。SCSIバスでは終端抵抗に供給する電源(TERMPWR)はSCSIバス上のどの機器が出力してもよいようになっています。これによって，電源を供給できる機器が1つでも生きていればバスが正しく動くようになっているわけです。

SCSIの規格ではTERMPWRには最大0.8Aの電流を供給することができること（シングルエンド型の場合）と，なんらかの異常があっても1A以上の電流が流れないようにヒューズなどで保護すること，TERMPWRの供給源が複数であってもよいようにダイオードで逆流防止を行うことが要求されています。

CZ-6BS1のTERMPWR関係の回路は図5のようになっています。CZ-6BS1もTERMPWRを出力するように設計されており，ダイオードと1Aのヒューズを入れています。TERMPWRが供給されないとSCSIバスは正常に動作しませんが，ヒューズが切れても，他のSCSIデバイスからTERMPWRが供給されていると異常に気がつかない場合がありますので注意してください。

●図……5　CZ-6BS1 の TERMPWR 関係回路



●5 I/Oマップ

CZ-6BS1 の I/O アドレスマップを図 6 に示します。SCSI バスの詳細や SPC の具体的な使用方法については拙著『Inside X68000』で実例とともに説明しておりますので，参考にしてください。

●6 コネクタ信号配置

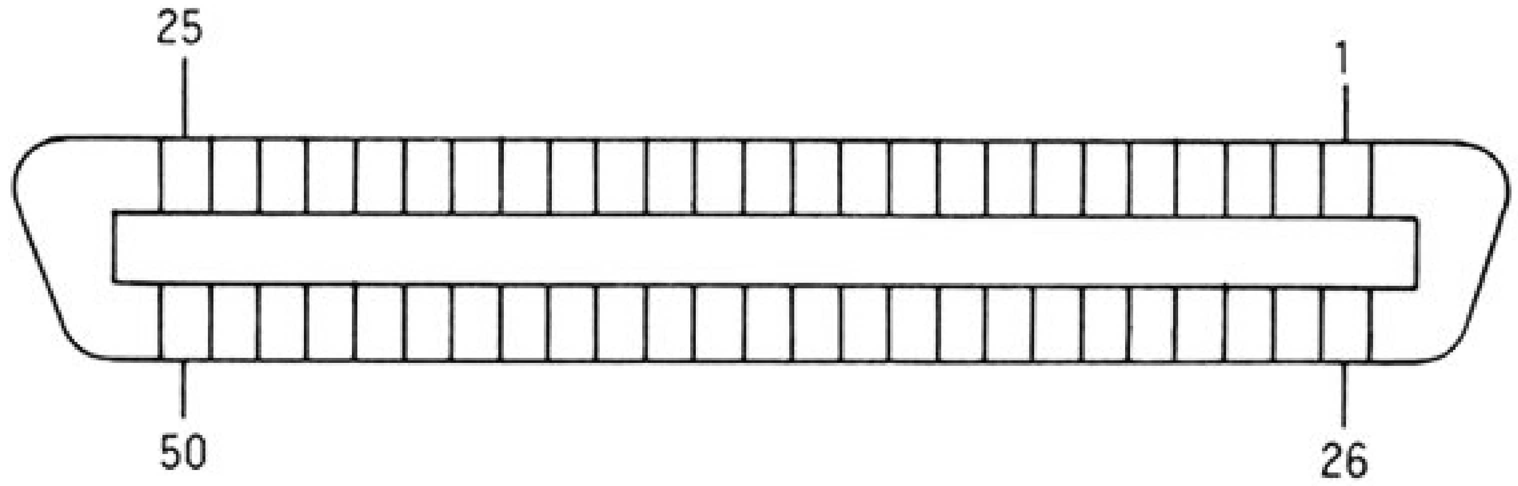
SCSI バスコネクタは，アンフェノールの 50 P コネクタを使用しています。コネクタの信号配置は図 7 のようになっています。

●図……6　SCSIボードのI/Oアドレスマップ

アドレス	READ/WRITE	bit 7	6	5	4	3	2	1	bit 0	レジスタ名称
+$1	R	ID #7	ID #6	ID #5	ID #4	ID #3	ID #2	ID #1	ID #0	BDID (Bus Device ID)
	W						ID			
+$3	R/W	Reset & Disable	Control Reset	Diag Mode	Arbitration Enable	Parity Enable	Select Enable	Reselect Enable	Interrupt Enable	SCTL (SPC Control)
+$5	R/W	Command Code			RST OUT	Intercept Transfer	Transfer Modifire			SCMD (SPC Command)
+$9	R	Selected	Reselected	Dis-Connected	Command Complete	Service Required	Timeout	SPC Hard Error	Reset Condition	INTS (Interrupt Sense)
	W	(Reset Interrupt:ビット配置はRead時と同じ)								
+$B	R	REQ	ACK	ATN	SEL	BSY	MSG	C/D	I/O	PSNS (Phase Sense)
	W	Diag REQ	Diag ACK	Xfer Enable		Diag BSY	Diag MSG	Diag C/D	Diag I/O	SDGC (SPC Diag Control)
+$D	R	Connected. INIT	TARG	SPC Busy	Transfer in Progress	SCSI Resetin	TC=0	DREG Status Full	Empty	SSTS (SPC Status)
+$F	R	Data Error SCSI	SPC	Xfer Out	'0'	TC Parity Error	'0'	Short Transfer Period	'0'	SERR (SPC Error Status)
+$11	R/W	Busfree INT Enable	'0'				Transfer Phase MSG	C/D	I/O	PCTL (Phase Control)
+$13	R	'0'				MBC				MBC (Modified Byte Counter)
+$15	R/W	Data								DREG (Data Register)
+$17	R	Temporary Data								TEMP (Temporary Register)
	W	Temporary Data								
+$19	R/W	Transfer Counter (上位)								TCH (Transfer Counter High)
+$1B	R/W	Transfer Counter (中位)								TCM (Transfer Counter Mid)
+$1D	R/W	Transfer Counter (下位)								TCL (Transfer Counter Low)

ベースアドレス:SCSIインタフェースボード(CZ-6BS1)……$EA0000
SCSI内蔵モデル　……$E96020

●図……7　SCSI バスコネクタの信号配置



ピン番号	信号名	ピン番号	信号名	機　能
1	GND	26	$\overline{DB0}$	データバスビット 0
2	GND	27	$\overline{DB1}$	〃　1
3	GND	28	$\overline{DB2}$	〃　2
4	GND	29	$\overline{DB3}$	〃　3
5	GND	30	$\overline{DB4}$	〃　4
6	GND	31	$\overline{DB5}$	〃　5
7	GND	32	$\overline{DB6}$	〃　6
8	GND	33	$\overline{DB7}$	〃　7
9	GND	34	$\overline{DBP}$	データバスパリティービット
10	GND	35	GND	
11	GND	36	GND	
12	GND	37	GND	
13	OPEN	38	TERMPWR	終端回路用電源
14	GND	39	GND	
15	GND	40	GND	
16	GND	41	$\overline{ATN}$	アテンション条件を示す信号
17	GND	42	GND	
18	GND	43	$\overline{BSY}$	バス使用中を示す信号
19	GND	44	$\overline{ACK}$	データ転送許可信号
20	GND	45	$\overline{RST}$	リセット信号
21	GND	46	$\overline{MSG}$	メッセージフェーズを示す信号
22	GND	47	$\overline{SEL}$	選択信号
23	GND	48	$C/\overline{D}$	コマンドかデータフェーズかを示す
24	GND	49	$\overline{REQ}$	データ転送要求信号
25	GND	50	$I/\overline{O}$	データの方向を示す信号

不平衡型（シングルエンド型）

●7 回路図

CZ-6BS1の回路図を図8に示します。

●図……8　CZ-6BS1の回路図（巻末参照）

GP-IBボード
(CZ-6BG1)

GP-IBは，計測機器と計算機をつなぐインタフェースとして一般的に使用されているものです。研究室で測定などの自動化を行うときにはなくてはならないものといえるでしょう。

GP-IBボードは計測機器のインタフェースとして普及しているIEEE std 488バス（GP-IB）をX68000シリーズでサポートするためのインタフェースボードです。

1　仕　様

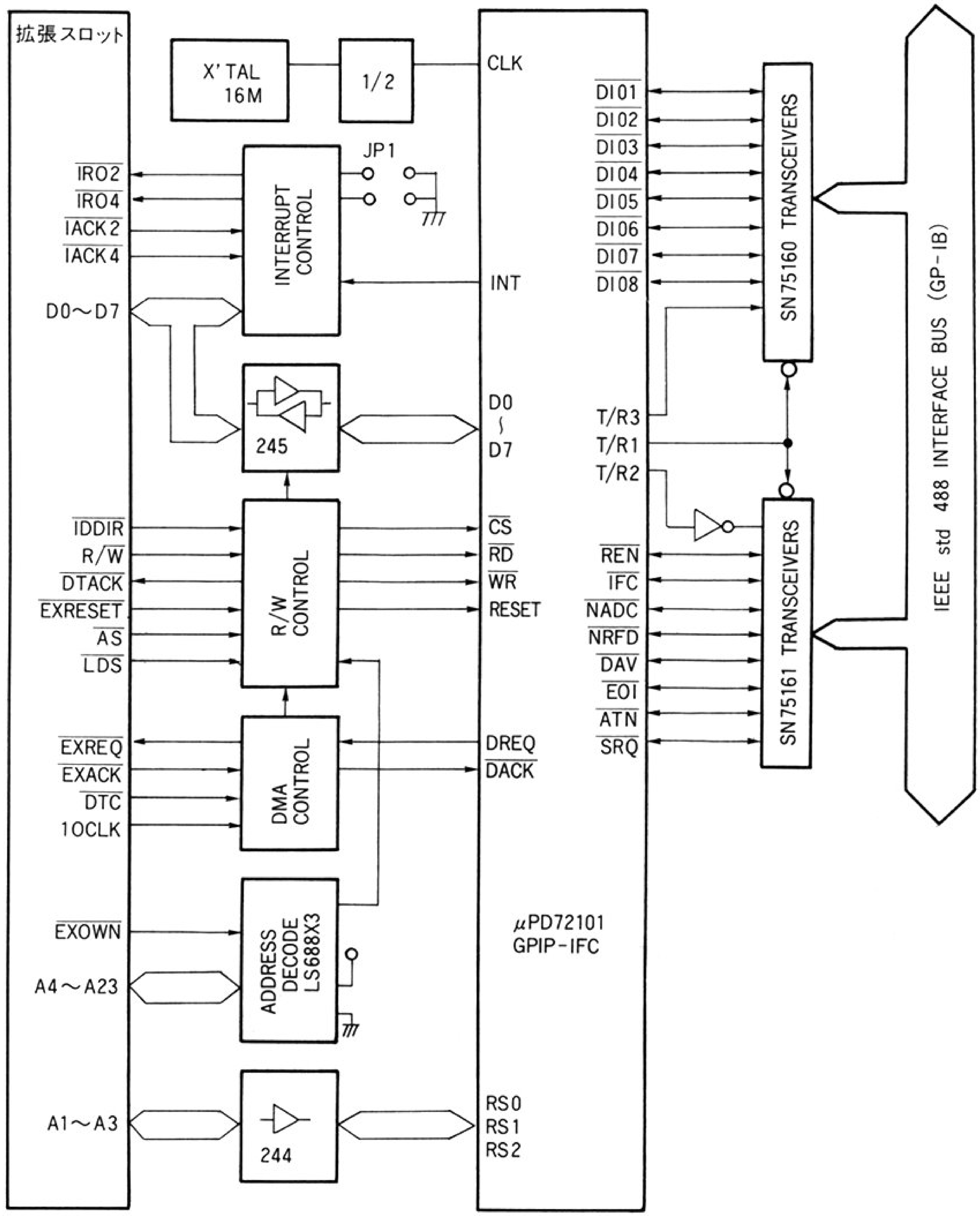
主要LSI	
伝送制御LSI	μPD 7210（NEC製）
GP-IBインタフェース	
準拠規格	GP-IB（IEEE std，488-1978）
チャンネル数	1
転送速度	50 KB/S MAX（プログラム転送時）
	400 KB/S MAX（DMA転送時）
サポートファンクション	SH 1（送信機能すべて）
	AH 1（受信機能すべて）

		T 6 (基本トーカ機能)
		L 4 (基本リスナ機能)
		SR 1 (サービスリクエスト機能)
		RL 1 (リモート/ローカル機能)
		DC 1 (デバイスクリア機能)
		DT 1 (デバイストリガ機能)
		C 1 (システムコントローラ機能)
		C 2 (IFC 送信機能)
		C 3 (REN 送信機能)
		C 4 (SRQ 送信機能)
		C 5 (I/F メッセージ機能)
	未サポートファンクション	TE0 (拡張トーカ機能)
		LE0 (拡張リスナ機能)
		PP0 (パラレルポール機能)
I/O コネクタ		24 ピンコネクタ
		57 LE-20240-770D 35G (DDK 社)
電源		
	電源電圧	+5 V
	消費電流	465 mA (TYP)
		430 mA (MIN)
		850 mA (MAX)
動作温度範囲		10～35℃
ポートアドレス		$EAFE 00～$EAFE 1 F
割り込み		レベル 2/レベル 4 (ジャンパースイッチで選択)
使用 DMA チャンネル		チャンネル# 2

●2 回路概要

CZ-6BG1 のブロック図を図 1 に示します。GP-IB コントローラ (μPD 7210) からトランシーバ IC (SN 75160/75161) を介して外部バスと接続されています。

●図……1　CZ-6BG1のブロック図

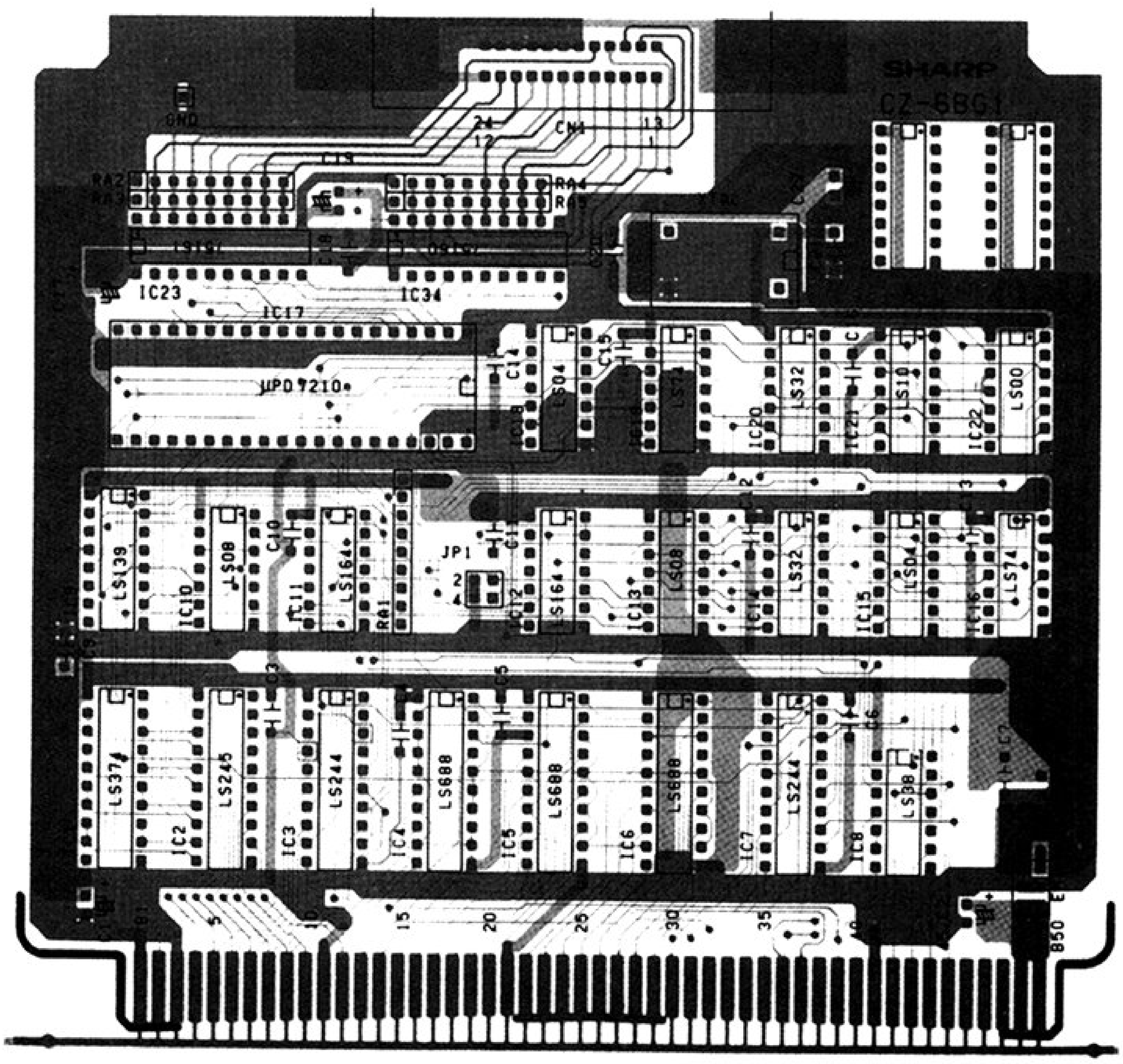


GP-IBのバス動作に関係するものはほとんどこのLSIが処理するため，CZ-6BG1の回路構成はX68000の拡張バスにμPD 7210を接続するための制御回路と，割り込みベクタの発生回路があるだけとなっています。

●3 主要部品配置

CZ-6BG1の部品配置を図2に示します。

●図……2　CZ-6BG1の部品配置



4 設　定

4・1 割り込み

CZ-6BG1 はジャンパースイッチ (JP1) によって，使用する割り込みレベルを選択することができるようになっています。2 側をショートするとレベル 2，4 側をショートするとレベル 4 の割り込みを使用するようになります。工場出荷時はレベル 2 を使用するように設定されています。

5 I/Oマップ

CZ-6BG1 の I/O アドレスマップを図 3 に示します。$EAFE01～$EAFE0F には GP-IB コントローラ LSI が，$EAFE11～$EAFE1F には割り込みベクタ番号レジスタが割り当てられています。μPD 7210 の使用方法を理解するには，GP-IB バスそのものの動作の詳細を知る必要があり，かなり面倒なので具体的な使用法については説明は省略します。

割り込みベクタ番号レジスタはアドレス空間上は 16 バイト分ありますが，実際に存在するのは 8 ビット分 1 つだけで，$EAFE11～$EAFE1F のどの奇数番地をアクセスしても同じものがアクセスされます。ソフト作成時にはベクタ番号レジスタは$EAFE11 にあるものとして作っておけばよいでしょう。

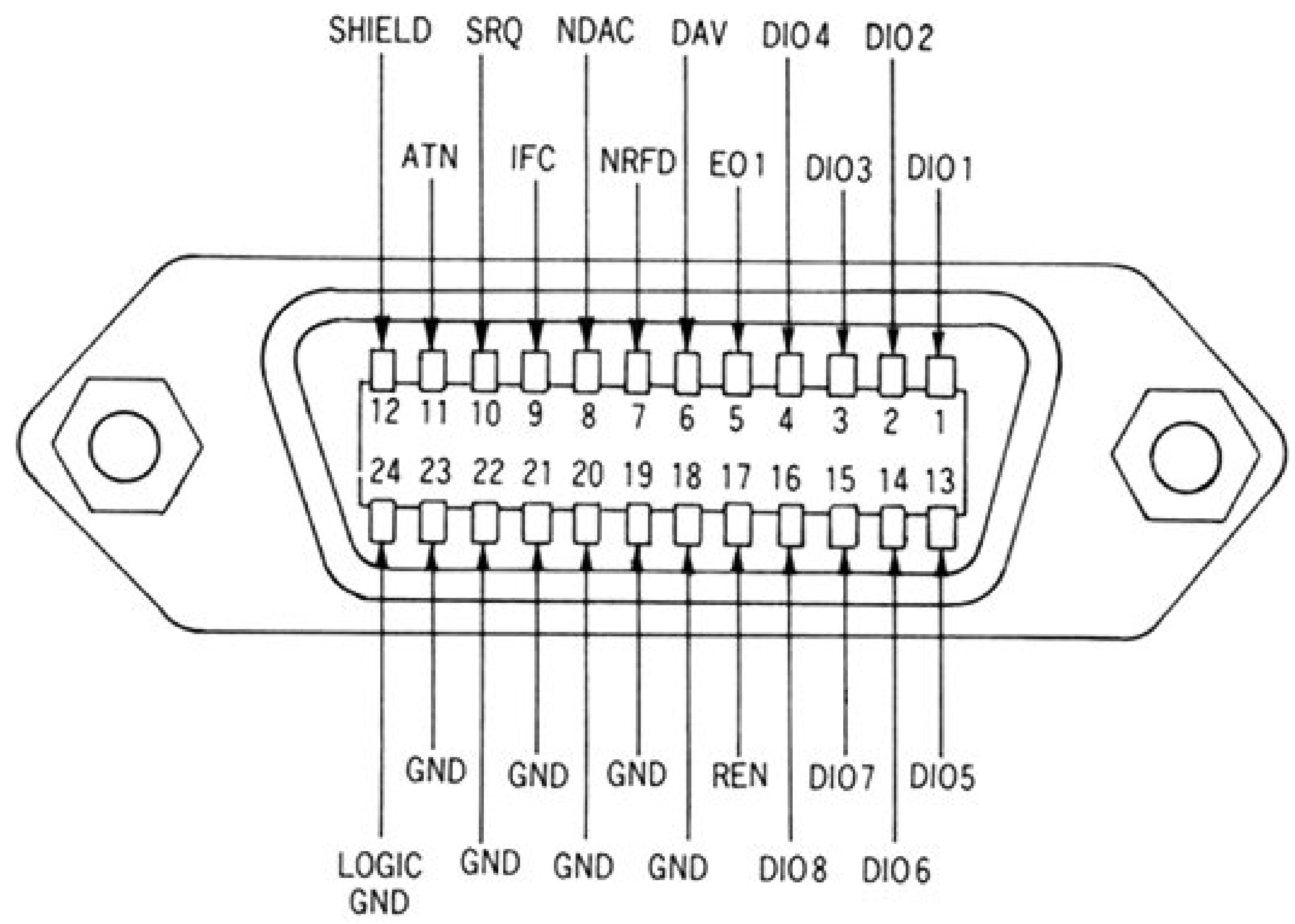
●図……3　GP-IG ボードのI/O アドレスマップ

デバイス		アドレス	レジスタ		7	6	5	4	3	2	1	0
μPD7210	READ	$EAFE01	0R	Data In	DI7	DI6	DI5	DI4	DI3	DI2	DI1	DI0
		$EAFE03	1R	Interrupt Status 1	CPT	APT	DET	END	DEC	ERR	DO	DI
		$EAFE05	2R	Interrupt Status 2	INT	SRQI	LOK	REM	CO	LOKC	REMC	ADSC
		$EAFE07	3R	Serial Poll Status	S8	PEND	S6	S5	S4	S3	S2	S1
		$EAFE09	4R	Address Status	CIC	ATN	SPMS	LPAS	TPAS	LA	TA	MJMN
		$EAFE0B	5R	Command Pass Through	CPT7	CPT6	CPT5	CPT4	CPT3	CPT2	CPT1	CPT0
		$EAFE0D	6R	Address 0		DT0	DL0	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0
		$EAFE0F	7R	Address 1	EO1	DT1	DL1	AD5-0	AD4-0	AD3-0	AD2-0	AD1-0
	WRITE	$EAFE01	0W	Byte Out	BO7	BO6	BO5	BO4	BO3	BO2	BO1	BO0
		$EAFE03	1W	Interrupt Mask 1	CPT	APT	DET	END	DEC	ERR	DO	DI
		$EAFE05	2W	Interrupt Mask 2	'0'	SRQI	DMAO	DMAI	CO	LOKC	REMC	ADSC
		$EAFE07	3W	Serial Poll Mode	S8	rsv	S6	S5	S4	S3	S2	S1
		$EAFE09	4W	Address Mode	ton	lon	TRM1	TRM0	'0'	'0'	ADM1	ADM0
		$EAFE0B	5W	Auxiliary Mode	CNT2	CNT1	CNT0	COM4	COM3	COM2	COM1	COM0
		$EAFE0D	6W	Address 0/1	ARS	DT	DL	AD5	AD4	AD3	AD2	AD1
		$EAFE0F	7W	End of string	EC7	EC6	EC5	EC4	EC3	EC2	EC1	EC0
		$EAFE11～$EAFE1F	割り込みベクタ番号		Interrupt Vector							

6　コネクタ信号配置

CZ-6BG1のコネクタの信号配置を図4に示します。コネクタの物理的な形状や信号配置はIEEE std, 488-1978に従ったものとなっています。

●図……4　GP-IB ボードのコネクタ信号配置



	バス構成信号線	備　考	
データ・バス	DIO 1 (Data Input/Output 1)	データ伝達をする	
	2 (Data Input/Output 2)	＜例＞　アドレス	
	3 (Data Input/Output 3)	コマンド	
	4 (Data Input/Output 4)	測定データ	
	5 (Data Input/Output 5)	プログラムデータ	
	6 (Data Input/Output 6)	表示データ	
	7 (Data Input/Output 7)	ステータス	
	8 (Data Input/Output 8)		
転送バス	DAV (Data Valid)	データの有効性を示す信号	アクセプタおよびソースハンドシェークを行う
	NRFD (Not Raedy For Data)	受信準備完了信号	
	NDAC (Not Data Accepted)	受信完了信号	
	ATN (Attention)	データ・バス上のデータがアドレスあるいはコマンドであることを示す信号	
	IFC (Interface Clear)	インタフェースを初期状態にする信号	
	SRQ (Service Request)	サービスを要求する信号	
	REN (Remote Enable)	リモート/ローカル指定信号	
	EOI (End or Identify)	データの最終バイトを示す。あるいはパラレル・ポールの実行を示す	

●7 回路図

CZ-6BG1 の回路図を図 5 に示します。

●図……5　CZ-6BG1 の回路図（巻末参照）

RS-232Cボード (CZ-6BF1)

RS-232Cは比較的低速なデータ転送用のインタフェースとして，ノートブック型のパソコンでも標準装備されるほど一般的なものとなっています。RS-232CボードはX68000に2チャンネルのRS-232Cポートを増設するものです。

RS-232 C ボードは 1 ボードあたり 2 チャンネルの RS-232 C ポートを X68000 に増設するものです。本体に付属する RS-232 C インタフェースと同様 8530 SCC を使用しており，さまざまな伝送モードに対応できるようになっているほか，Aチャンネルは DMA による高速伝送，Bチャンネルは外部に電源（+5 V/+12 V/−12 V）を供給することができるようになっており，光モデムなどを接続するときに外部電源を省略できるようになっています。

1 仕　様

主要 LSI	
伝送制御 LSI	LH8530-SCC
シリアルインタフェース	
準拠規格	RS-232 C（EIA），JIS-X 5101
チャンネル数	2
通信方式	同期式，調歩同期式

ボーレート	75～19200 bps
キャラクタ長	5/6/7/8
ストップビット長	1/1.5/2
パリティビット長	なし/偶数/奇数
I/O コネクタ	25 ピン D-SUB コネクタ
電源	
電源電圧	+5 V/+12 V/−12 V
消費電流（外部供給分を除く）	
+5 V	370 mA
+12 V	35 mA
−12 V	50 mA
動作温度範囲	10～30℃
ポートアドレス	$EAFC01～$EAFC09/$EAFC11～$EAFC19 $EAFC21～$EAFC29/$EAFC31～$EAFC39 の 4 種類のうち 1 つ（ジャンパースイッチで選択）
割り込み	レベル 2/レベル 4（ジャンパースイッチで選択）

●2 回路概要

CZ-6BF1 のブロック図を図 1 に示します。X68000 の本体の RS-232C インタフェースに使用されているのと同じ LH8530SCC が 1 個使用されています。SCC はA，B 2 つのチャンネルを持っていますので，CZ-6BF1 も 2 つの RS-232 C コネクタを設けています。

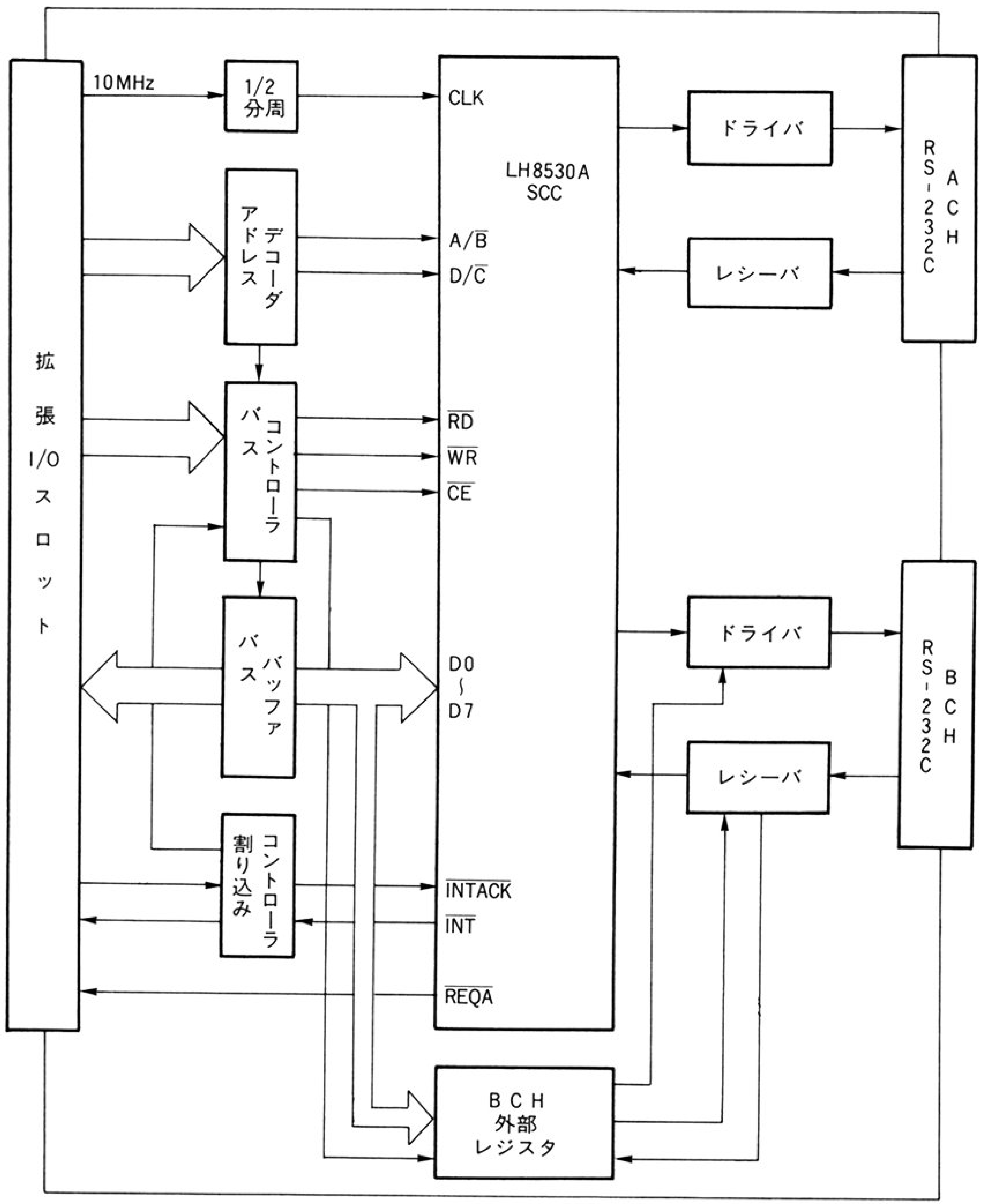
CZ-6BF1 では，SCC の持つ RS-232C 制御信号のうちAチャンネルの DCD に RS-232C の DSR 信号を接続し，Bチャンネル用の DTR，CTS，DCD 端子をそれぞれAチャンネルの送信クロック源の切り替え（SCC の内部で作るか，ST 2 を使用するか）出力，CI 信号，CD 信号の入力に流用しています。

このためBチャンネルのクロック切り替え，DTR，CTS，DSR 信号などが不足します。CZ-6BF1 では別途 I/O ポートを増設し，これによってBチャンネルのクロック源切り替えや DTR 出力，CTS や DSR 信号の状態読み出しを行うようにしています。

RS-232C 規格を重視したAチャンネル，シリアル伝送を使ったユニークな周辺機器を作成，

接続するのに便利なＢチャンネルとみることもできるでしょう。

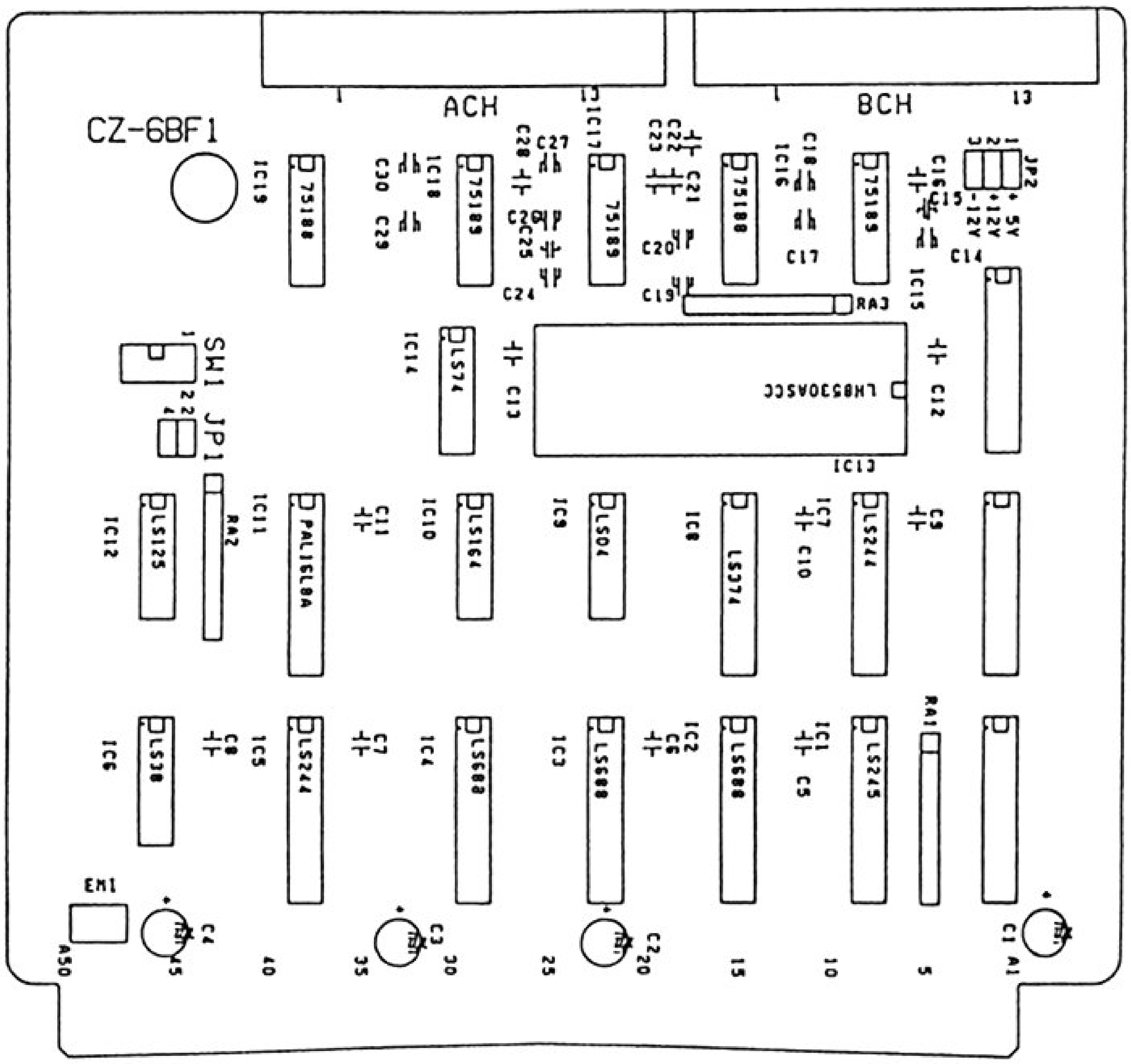
●図……1　CZ-6BF1 のブロック図



●3 主要部品配置

CZ-6BF1 の部品配置を図 2 に示します。

●図……2　CZ-6BF1 の部品配置

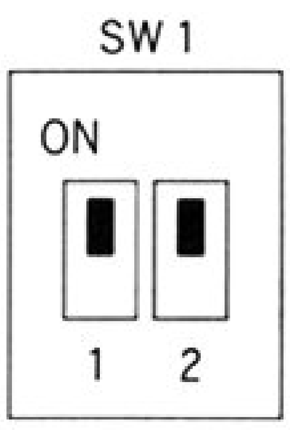


●4 設　定

❹·1 ボードアドレスの設定

CZ-6BF1 はボード上のスイッチ（SW 1）によって，I/O ポートアドレスを変更することができるようになっています（図3）。

●図……3　ディップスイッチ（SW 1）



ONの時“0”
OFFの時“1”

左の例は，工場出荷時の設定です。

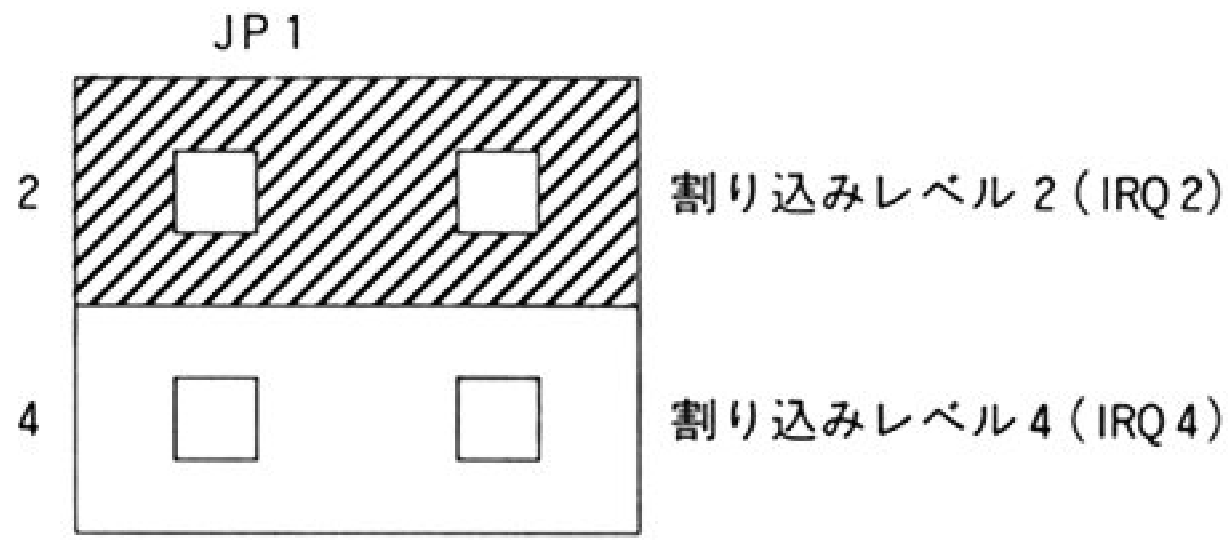
SW 1		ボード番号	I/O ポートアドレス
1	2		
ON	ON	0（1枚目）	$EAFC01～$EAFC09
OFF	ON	1（2枚目）	$EAFC11～$EAFC19
ON	OFF	1（3枚目）	$EAFC21～$EAFC29
OFF	OFF	2（4枚目）	$EAFC31～$EAFC39

SW 1の 2 ビットのスイッチが両方とも ON なら$EAFC01～$EAFC09，1 が OFF で 2 が ON なら$EAFC11～$EAFC19，1 が ON で 2 が OFF なら$EAFC21～$EAFC29，両方とも OFF なら$EAFC31～$EAFC39 を使用するようになります。

❹·2 割り込み

ジャンパースイッチ（JP 1）の設定によって使用する割り込みレベルを選択することができます（図4）。ショートピンを 2 番に差すと IRQ 2 が，4 番に差すと IRQ 4 が使用されるようになります。

●図……4　ジャンパースイッチ（JP 1）



●5 I/Oマップ

CZ-6BF1 の I/O アドレスマップを図 5 に示します。LH8530（SCC）の使用方法については拙著『Inside X68000』でも解説していますので参考にしてください。

●図……5　RS-232C ボードの I/O アドレスマップ

デバイス	アドレス	レジスタ	7	6	5	4	3	2	1	0
LH8530	ベースアドレス ＋$01	コマンドレジスタ Bポート								
	＋$03	データレジスタ Bポート								
	＋$05	コマンドレジスタ Aポート								
	＋$07	データレジスタ Aポート								
LS374	WRITE ＋$09	外部Bチャンネル 書き込みレジスタ							DSR	STB
LS244	READ	外部Bチャンネル 読み出しレジスタ					DTR	STB	CTS	DSR

ベースアドレス：$EAFC00（1枚目）/$EAFC10（2枚目）/$EAFC20（3枚目）/$EAFC30（4枚目）

●外部Bチャンネル書き込みレジスタ

bit 7	6	5	4	3	2	1	bit 0
						DTR	STB

STB：同期通信時の送信タイミングクロック源選択
'1':SCC内部で作ったクロックを使用する
'0':ST2入力を使用する（ST1＝ST2となる）

DTR：チャンネルBのDTR信号の状態を設定する
'1':DTR端子をOFF状態にする
'0':DTR端子をON状態にする

●外部Bチャンネル読み出しレジスタ

bit 7	6	5	4	3	2	1	bit 0
				DTR	STB	CTS	DSR

DSR：チャンネルBのDSR信号の状態を示す
'1':DSRはOFF状態
'0':DSRはON状態

CTS：チャンネルBのCTS信号の状態を示す
'1':CTSはOFF状態
'0':CTSはON状態

STB：チャンネルB書き込みレジスタのSTBビットの状態を示す

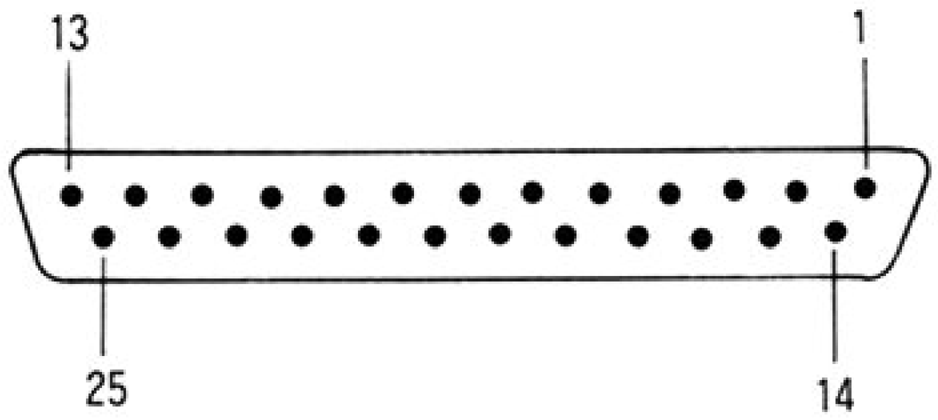
DTR：チャンネルB書き込みレジスタのDTRビットの状態を示す

●6 コネクタ信号配置

CZ-6BF1のコネクタ信号配置を図6に示します。CZ-6BF1は2チャンネルのRS-232Cコネクタを持っていますが，チャンネルBはCD，CI信号を持っていないことと，Bチャンネルは規格上未接続になっているピンから電源（+5 V, +12 V, −12 V）を出力できるようにしている点に気をつけてください。

他の機器と接続するケーブルを自作するときには，電源出力が行われるおそれのあるピン（9，10，14番ピン）はオープンにしておく方がよいでしょう。

●図……6　RS-232C ボードのコネクタ信号配置



（ケーブル接続側より見た図）

端子番号	CZ-6BF1 信号名	機能	方向 DTE/DOE	各規格略号 JIS	EIA	CCITT	備考
1	FG	保安用アース	←→	SG	AB	102	
2	T×D	送信データ	→	SD	BA	103	出力
3	R×D	受信データ	←	RD	BB	104	入力
4	RTS	送信要求 ONにより相手を送信可能とする	→	RS	CA	105	出力
5	CTS	送信可 ONの時送信可能と判断する	←	CS	CB	106	入力
6	DSR	データセットレディ ONの時，相手準備完了状態	←	DR	CC	107	入力
7	SG	信号用アース	←→	SG	AB	102	
8	CD	キャリア検出 ONの時，受信データ有効	←	CD	CF	109	※1 入力
9	+12V	DC+12V	→	—	—	—	※2 出力
10	−12V	DC−12V	→	—	—	—	※2 出力
14	+5V	DC+5V	→	—	—	—	※2 出力
15	ST2	送信信号エレメントタイミング 同期式送信タイミング	←	ST2	DB	114	入力
17	RT	受信信号エレメントタイミング 同期式受信タイミング	←	RT	DD	115	入力
20	DTR	データ端末レディ ONで当方準備完了	→	ER	CD	108/2	出力
22	CI	被呼表示 ONで相手から呼び出しあり	←	CI	CE	125	※1 入力
24	ST1	送信信号エレメントタイミング 同期式送信タイミング	→	ST1	DA	113	出力

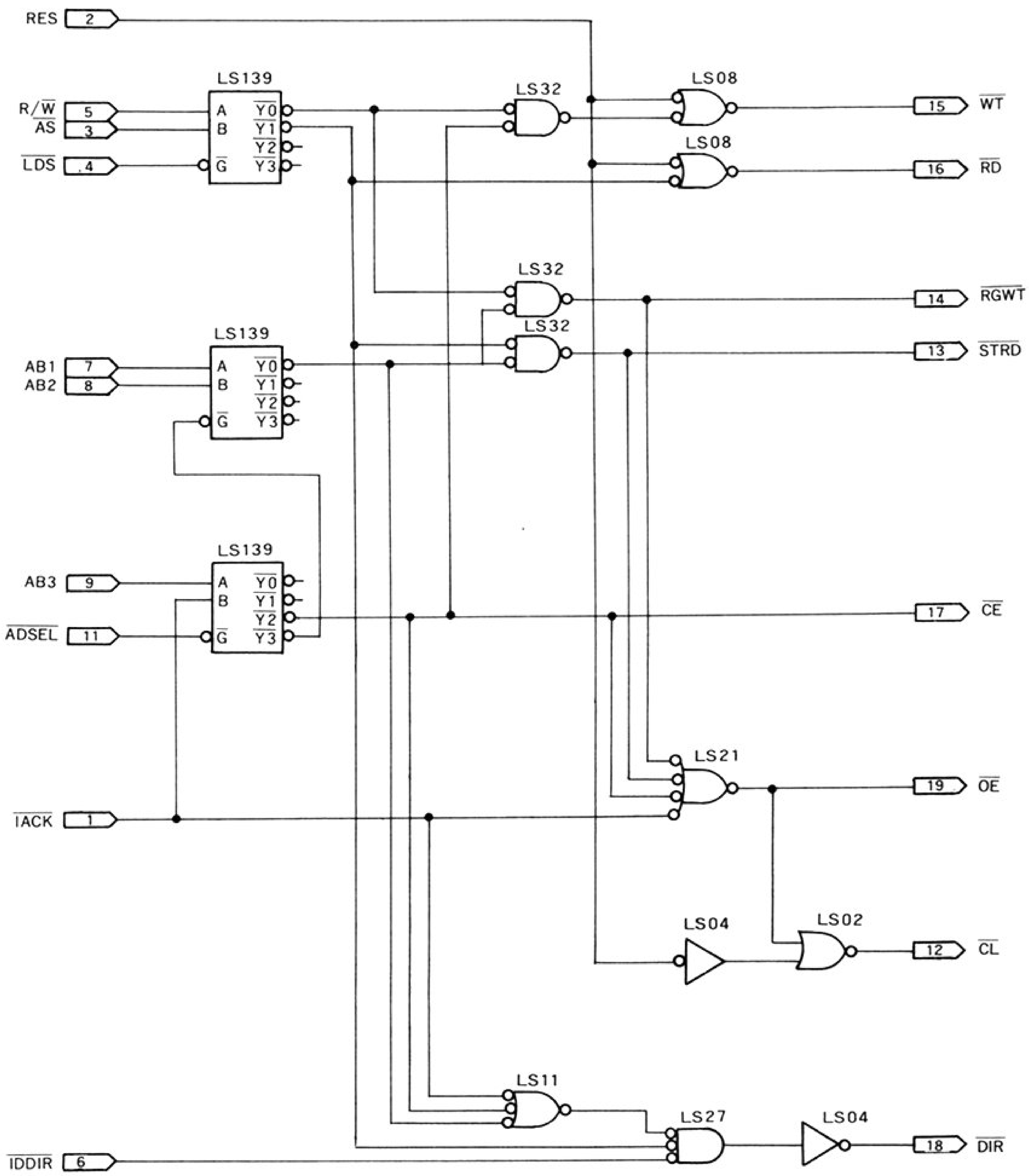
※1　Bチャンネルは未接続です。
※2　BチャンネルはJP2を通して接続することができます。
　　Aチャンネルは未接続です。

●7 回路図

CZ-6BF1 の回路図を図 7 に, 基板上にある PAL の等価回路を図 8 に示しますので, 参考にしてください。

●図……7　CZ-6BF1 の回路図 (巻末参照)

●図……8　PAL の等価回路


RES
2
R/W
5
AS
3
LDS
4
LS139
A
B
G
Y0
Y1
Y2
Y3
LS32
LS08
15
WT
LS08
16
RD
LS32
14
RGWT
LS32
13
STRD
AB1
7
AB2
8
LS139
AB3
9
ADSEL
11
LS139
17
CE
LS21
19
OE
IACK
1
LS04
LS02
12
CL
LS11
LS27
LS04
18
DIR
IDDIR
6

FAXボード
(CZ-6BC1)

ファクシミリによる画像転送は，文字だけでなく，グラフィック画面なども画質の劣化はほとんど起こさずに送ることができます。FAXボードはX68000から直接公衆回線と接続し，FAX通信を行うものです。

FAXボード（CZ-6BC1）は，電話回線を利用して4800 bpsの半二重データ通信（CCITT V 27 ter, V 21チャンネル2準拠）を行うことができるモデムボードです。ボードに付属するソフトウェア「FAXツール」を使用することで，G 3規格のファクシミリ通信を行うことができます。

1 仕様

主要LSI

モデムLSI	R48MFX
ステータス入力/制御	82C55
オプション	
通信制御LSI	Z8530（SCC）
9600 bpsモデムLSI	R96MD

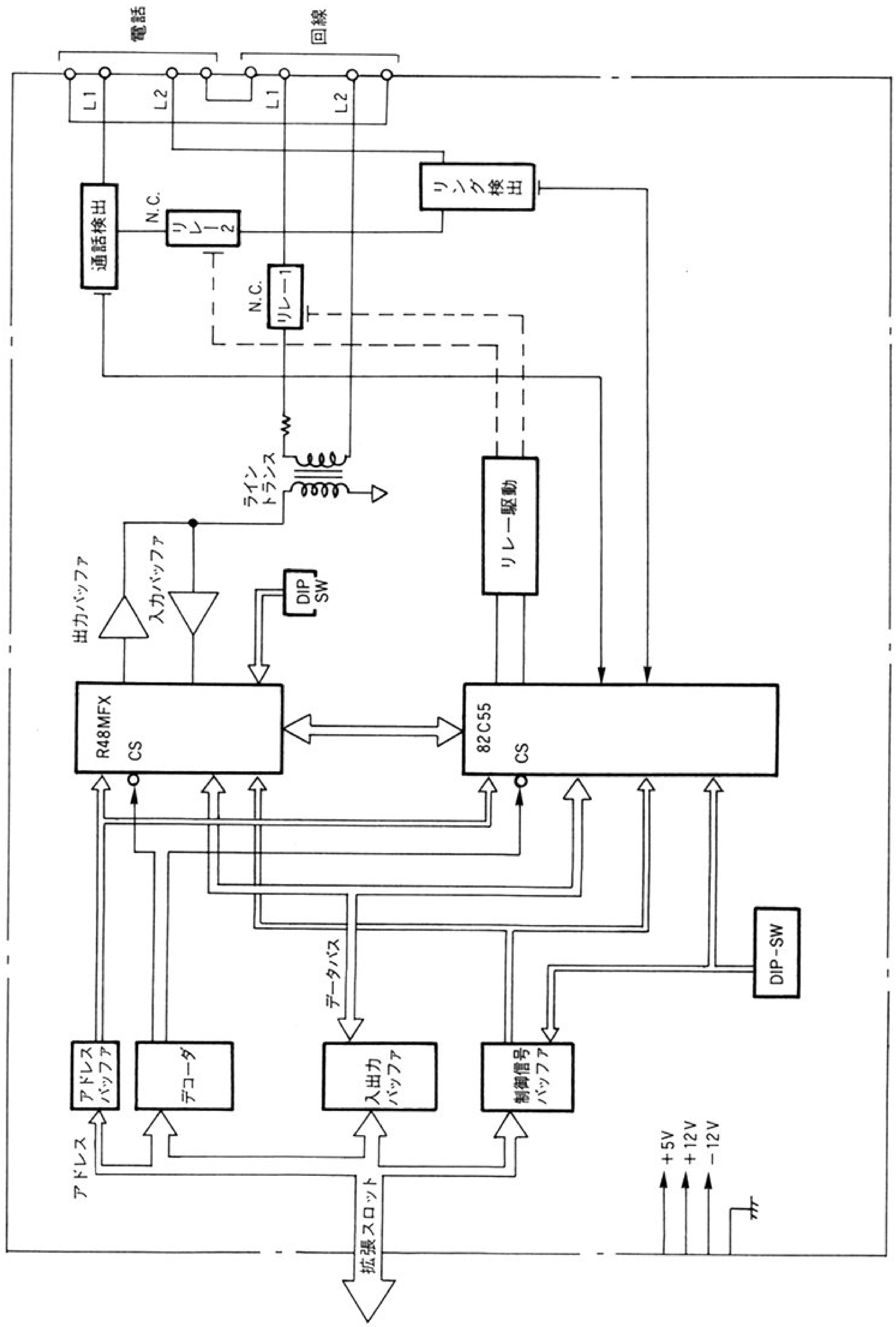
回線	
適用回線	一般公衆電話回線，PBX 内線
回線接続方式	通信コネクタ（モジュラージャック）
NCU 部	AA 型（ソフトウェアコントロール）
選択信号種別	DP（10 PPS/20 PPS），PB
モデム	
通信速度	300/2400/4800 bps
通信方式	V 21 チャンネル 2，V 27 ter
送信信号レベル	－6 から－15 dBm まで 3 dBm ステップで選択可（出荷時：－15 dBm）
電源	
電源電圧/消費電流	＋5 V　約 500 mA ＋12 V 約 10 mA －12 V 約 25 mA
動作温度範囲	
ポートアドレス	$EAF900～$EAF95F （8255　：$EAF 900～$EAF 907） （Z 8530　：$EAF 908～$EAF 90 B）※オプション （R 48 MFX ：$EAF 920～$EAF 93 F） （R 96 MD　：$EAF 940～$EAF 95 F）※オプション
割り込み	レベル 2/レベル 4（DIP スイッチで選択）

●2 回路概要

CZ-6BC1 のブロック図を図 1 に示します。モデム IC として R48MFX が，DCD や CTS などのステータスチェックや回線用のリレー制御などのために 82C55(8255 の C-MOS 版。ソフト的には 8255 と同じ) が使用されています。

電話回線と X68000 との間は直接(DC 的に)接続されないよう，データラインはライントランス経由で，リング (呼鳴) 検出や通話検出 (電話を使用中か否かを判別する) 部と 8255 の間はフォトカプラによって絶縁されています。

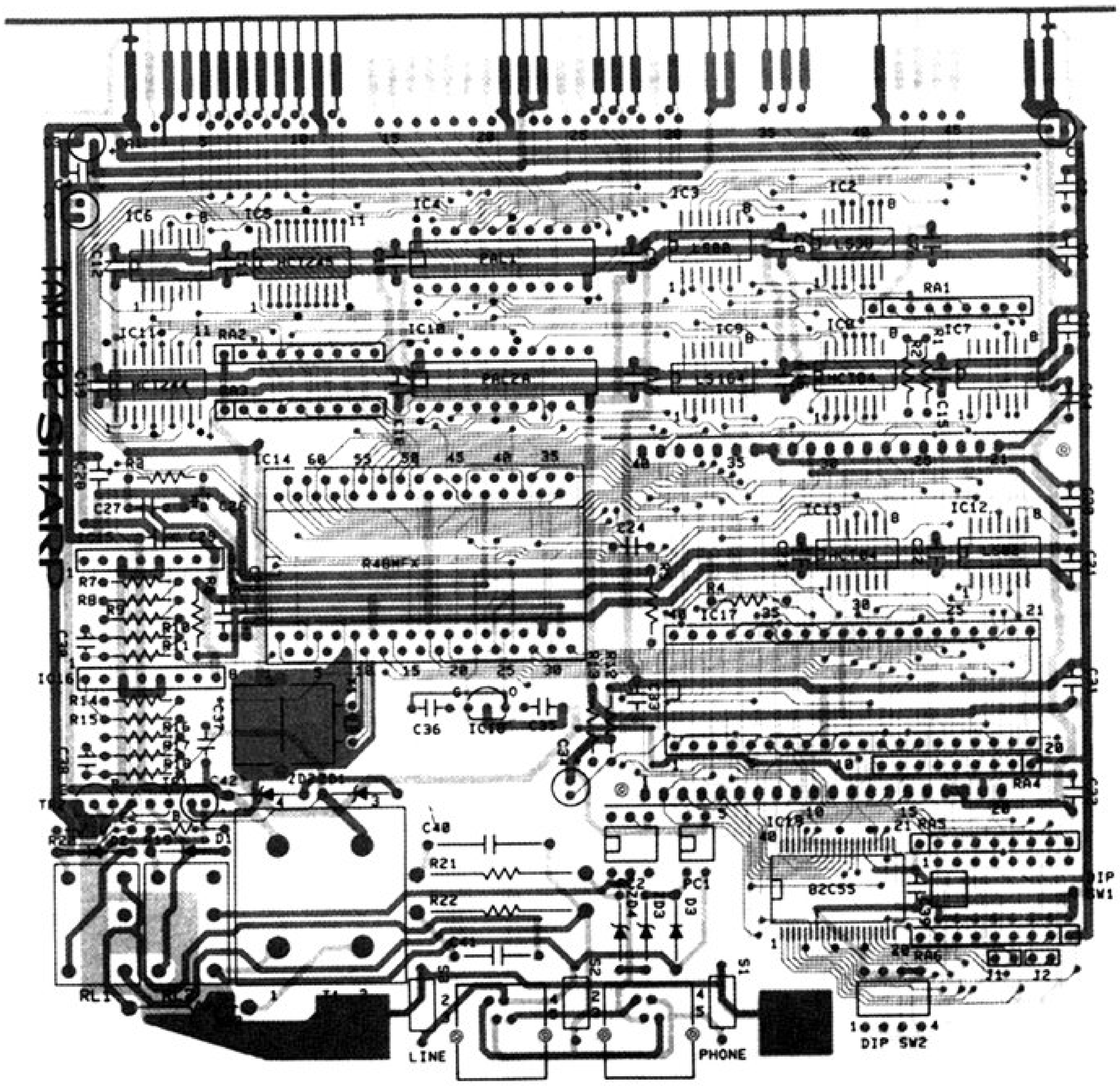
●図……1　CZ-6BC1 のブロック図



●3 主要部品配置

CZ-6BC1の部品配置を図2に示します。

●図……2　CZ-6BC1の部品配置



●4 設　定

CZ-6BC1には2つのディップスイッチがあります。それぞれの設定内容は図3のようになっています。

●図……3　CZ-6BC1のディップスイッチ

• DIP-SW1

No. 1	No. 2	送出レベル	
OFF	OFF	− 6dBm	
OFF	ON	− 9dBm	
ON	OFF	−12dBm	
ON	ON	−15dBm	出荷時の設定

• DIP-SW2

	出荷設定	設定内容
No. 1	ON	ケーブルイコライザ設定
No. 2	ON	
No. 3	ON	割り込みレベル設定
No. 3	OFF	

❹·1 回線送出レベルの設定

DIP-SW 1によって回線に送出する信号レベルの設定を行うことができます。回線の減衰量(単位：dBm) と加算したレベルが−15 dBmを超えないように調整します。

この設定は，かならず電話工事のための資格を持った工事担任者が行うようにしてください。

❹·2 ケーブルイコライザの設定

DIP-SW 2の1，2によって回線の周波数特性の補償のためのイコライザの調整を行うことができます。

❹·3 割り込みレベルの設定

CZ-6BC1はボード上のDIPスイッチによって使用する割り込みレベルを選択することができるようになっています。DIP-SW 2の3がONになっているとレベル4が，DIP-SW 2の4がONになっているとレベル2の割り込みが使用されるようになります。両方ともONにすると割り込みが使用されなくなります。両方ともOFFに設定するとレベル2とレベル4の両方に割り込みが発生してしまいますので，このような設定は行わないようにしてください。

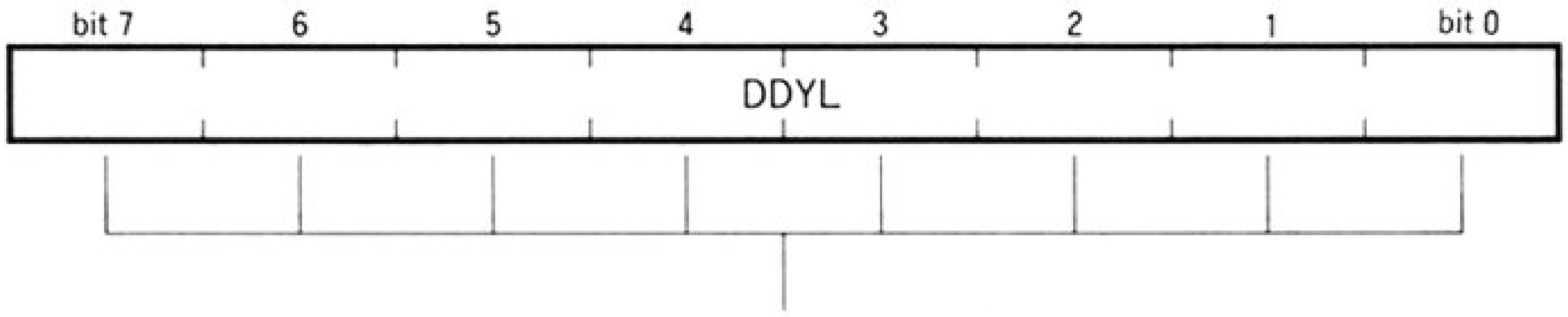
●5 I/Oマップ

CZ-6BC1のI/Oアドレスマップを図4に，各ビットの内容を図5～図21に示します。

●図……4　FAX ボードのI/Oアドレスマップ

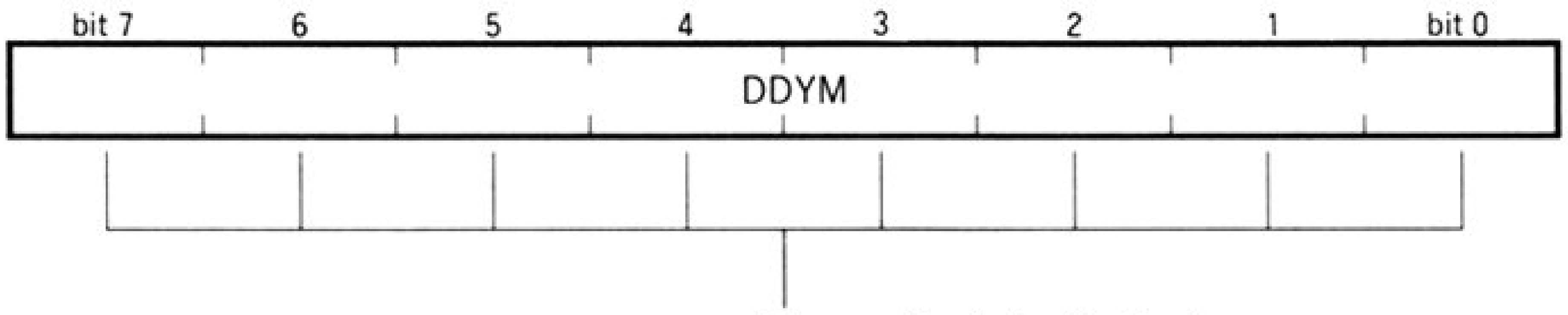
アドレス	レジスタ番号	ビット配置							
		7	6	5	4	3	2	1	0
$EAF921	0	DDYL							
$EAF923	1	DDYM							
$EAF925	2	DDXL							
$EAF927	3	DDXM							
$EAF929	4	TXCD							
$EAF92B	5	RXCD							
$EAF92D	6								
$EAF92F	7								
$EAF931	8			CDET		PN			
$EAF933	9								
$EAF935	A	TDET							
$EAF937	B	RX	FED		GHIT				
$EAF939	C	RTSP	EPT	TPDM	TDIS	EQSV	EQFZ	SDIS	RAMW
$EAF93B	D	CONF							
$EAF93D	E	LA	CDIE	CDREQ		SETUP	DDIE		DDAEQ
$EAF93F	F	RAMA							

●図……5　レジスタ 0



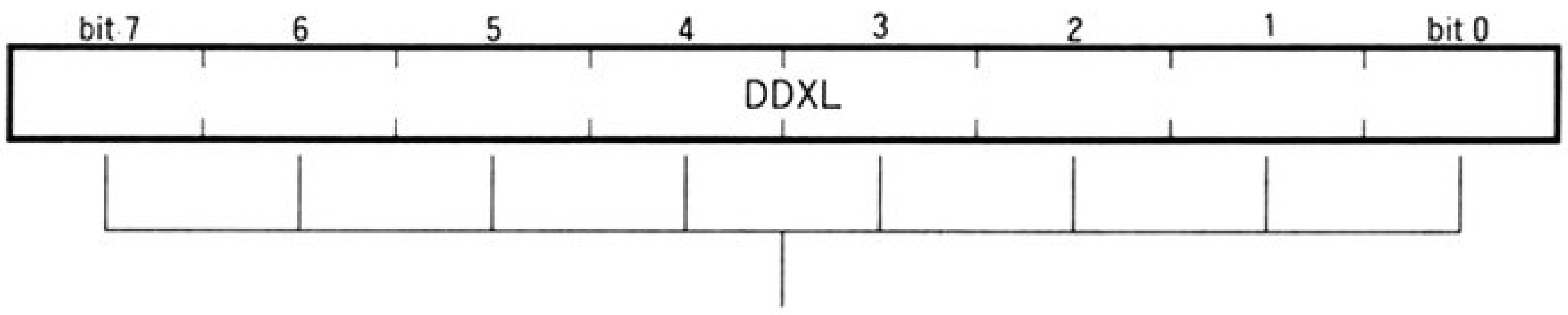
Diagnostic Data Y Least
YRAMロケーションの読み出しや，XRAM/YRAMへの書き込みを行うときに使用される16ビットデータの下位8ビット

●図……6　レジスタ 1



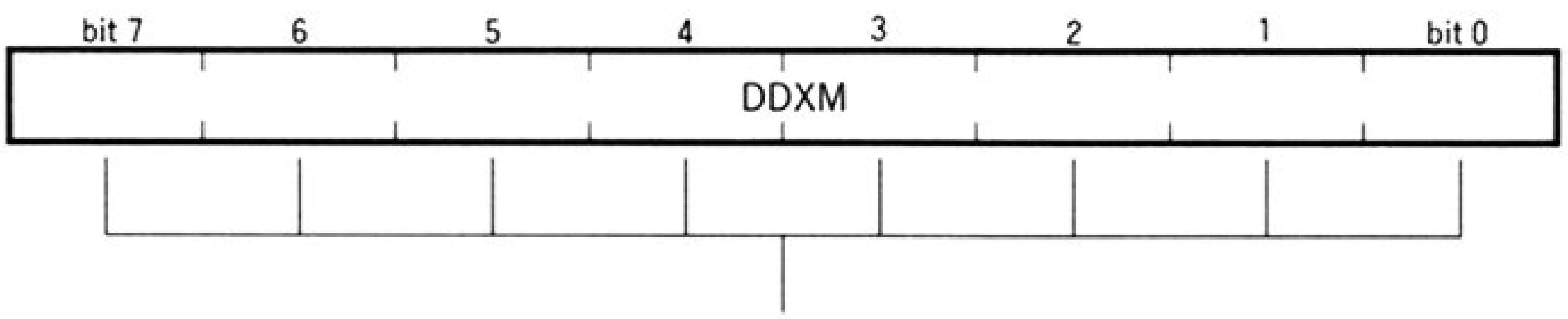
Diagnostic Data Y Most
YRAMロケーションの読み出しや，XRAM/YRAMへの書き込みを行うときに使用される16ビットデータの上位8ビット

●図……7　レジスタ 2



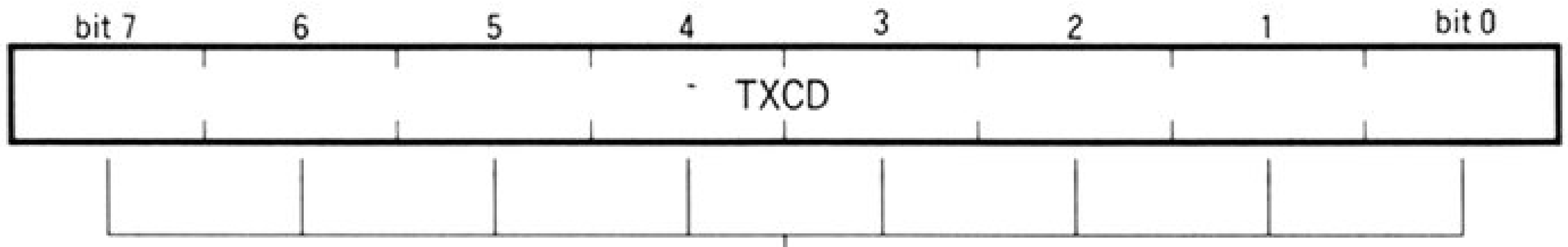
Diagnostic Data X Least
XRAMロケーションを読み出すときに使用される16ビットデータの下位8ビット

●図……8　レジスタ 3



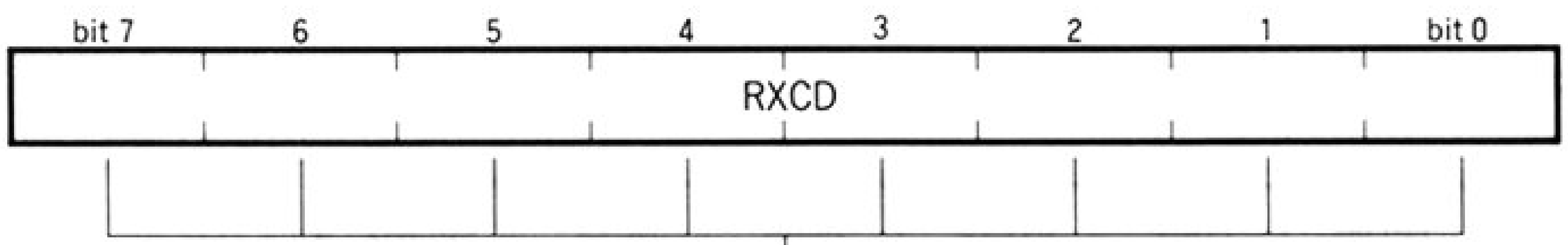
Diagnostic Data X Most
XRAMロケーションを読み出すときに使用される16ビットデータの上位8ビット

●図……9　レジスタ4



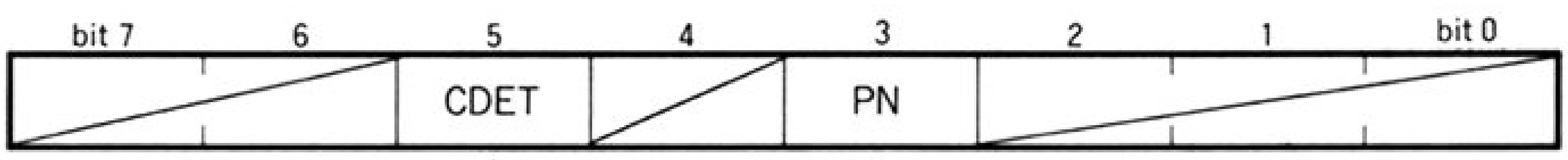
Transmitter Channel Data
CDREQ(チャンネルデータ要求ビット)が'1'のときにこのレジスタに書き込んだデータが送信器に送られる

●図……10　レジスタ5



Receiver Channel Data
モデムが8ビット受け取るごとに，CDREQビットを'1'にするとともにこのレジスタに受信したデータがセットされる

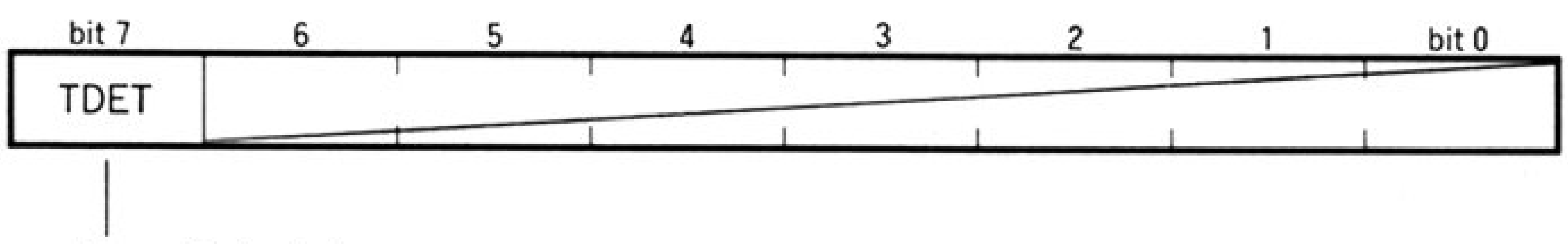
●図……11　レジスタ8



Period N
'1':受信シーケンスがスタートした
'0':受信スクランブル1が開始した
＊TDIS(レジスタCビット4)が'1'のとき，このビットは無効

Carrier Detector
バンドパスエネルギーが検出され，トレーニング・シーケンスではないことを示す
'1':データステート開始
'0':受信信号の終了

●図……12　レジスタA



Tone Detected
'1':トーンを受け取った
'0':トーンを受け取っていない

●図……13　レジスタB

bit 7	6	5	4	3	2	1	bit 0
RX	FED		GHIT				

Gain Hit
'1':AGC回路が修正可能とする以上の急激な速さで受信レベルが増加した
'0':AGC出力は通常状態に復帰した

Fast Energy Detector
受信信号のレベルを示す
'00':受信レベルなし
'01':無効
'10':ターンオフ・スレッショルド以上
'11':ターンオン・スレッショルド以上

●図……14　レジスタC

bit 7	6	5	4	3	2	1	bit 0
RTSP	EPT	TPDM	TDIS	EQSV	EQFZ	SDIS	RAMW

RAM Write
'1':モデムICに診断用書き込みを行う
'0':モデムICから診断用データを読み出す

Scrambler Disable
'1':スクランブラ／ディスクランブラ使用不可
'0':スクランブラ／ディスクランブラ使用可

Equalizer Freeze
'1':適応イコライザのタップ更新停止
'0':通常動作

Equalizer Save
'1':適応イコライザのタップはモデム再構成時やトレーニング時にゼロ化されない
'0':通常動作

Training Disable
'1':モデムをトレーニングフェーズに入れなくする（RTSやRTSPがアクティブになったとき，送信開始時にトレーニングシーケンスを発生することができなくなる）
'0':通常動作

Transmitter Parallel Data Mode
'1':TXCDレジスタの内容を送信する
'0':シリアルハードウエア・データビッドを送信する

Echo Protector Tone
'1':トレーニングシーケンスの初めで非変調搬送波が185mS送信された後，送信エネルギーのない状態が20mS続く（TDISが'1'の場合にはこのビットは'1'になりません）
'0':通常動作

Request Send Parallel
'1':送信シーケンスを開始する(RTSPが'0'になり，ターンオフ・シーケンスが終了するまで送信を継続する）
'0':　　　〃　　開始しない

●図……15 レジスタD

bit 7	6	5	4	3	2	1	bit 0
CONF							

Configuration
モデムのコンフィグレーションを行う
$00:V.21(300bps)
$04:V.27,2400 ロングトレーニング
$05:V.27,2400 ショートトレーニング
$06:V.27,4800 ロングトレーニング（デフォルト）
$07:V.27,4800 ショートトレーニング
$08:トーン(*1)
その他:無効

＊1: モデムはRTS，またはRTSP信号に対応して単独または二重周波数トーンを送り出します。トーンの周波数と振幅はホストによって書き込まれるRAMロケーションによってコントロールされます。
トーンを発信しない場合，トーン構成はTDETビットによる単独周波数トーンの検出を可能にします。
トーン・ディテクタ周波数はホストにより，いくつかのRAMロケーションを変更することにより変えることができます。

●図……16 レジスタE

bit 7	6	5	4	3	2	1	bit 0
IA	CDIE	CDREQ	／	SETUP	DDIE	／	DDREQ

Diagnostic Data Request
'1': モデムがDDYLにアクセスした
'0':ホストCPUがDDYLにアクセスした

Diagnostic Data Interrupt Enable
'1':DDREQ＝'1'で割り込みを発生する
'0': 〃 発生しない

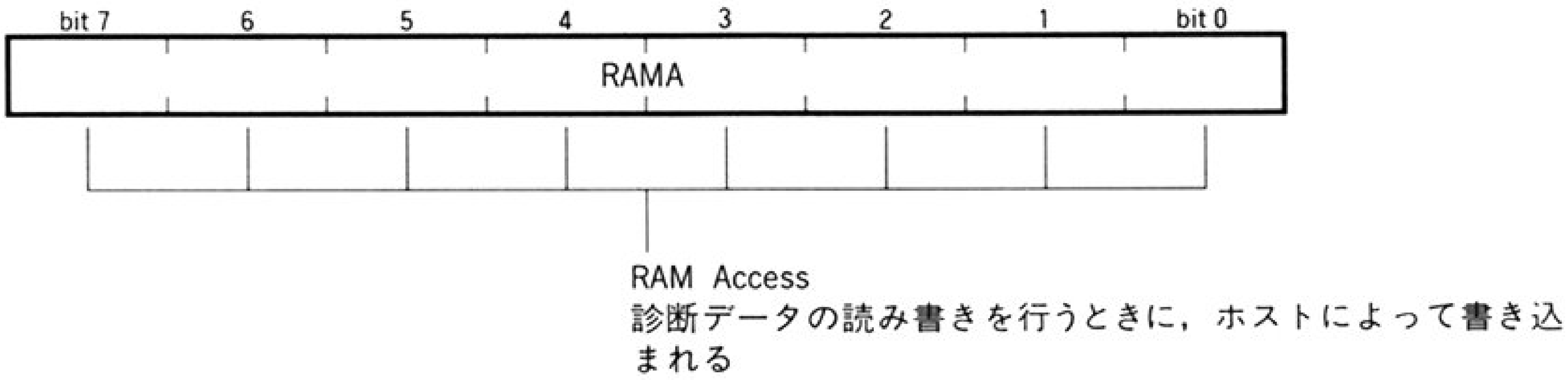
Setup
モデム再構成するとき(CONFレジスタを変更するとき)'1'にする

Channel Data Request
'1':モデム受信器がデータをRXCDに書き込むか，RXCDを読み出した
ホストプロセッサはサービス終了後このビットを'0'に戻す必要がある

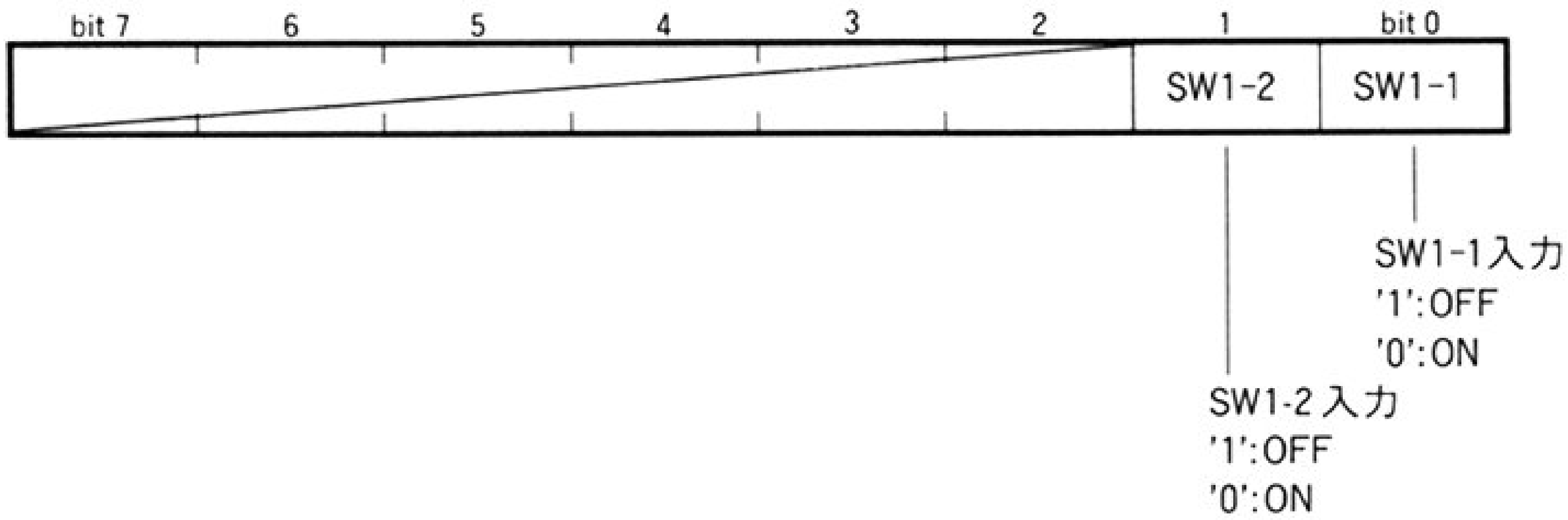
Channel Data Interrupt Enable
'1':CDREQ＝'1'で割り込みを発生する
'0': 〃 発生しない

Interrupt Active
'1':割り込み要求中
'0':通常動作

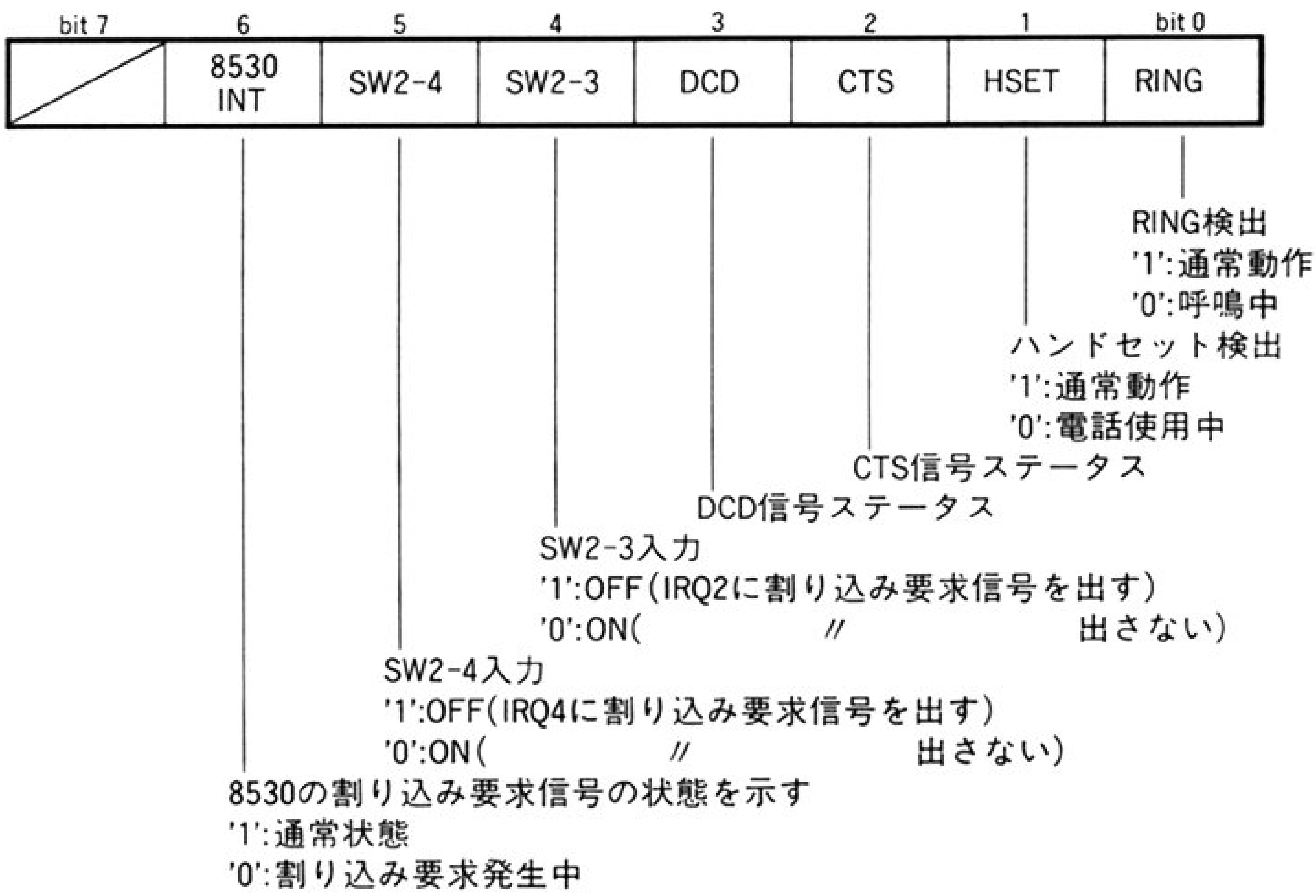
●図……17　レジスタF



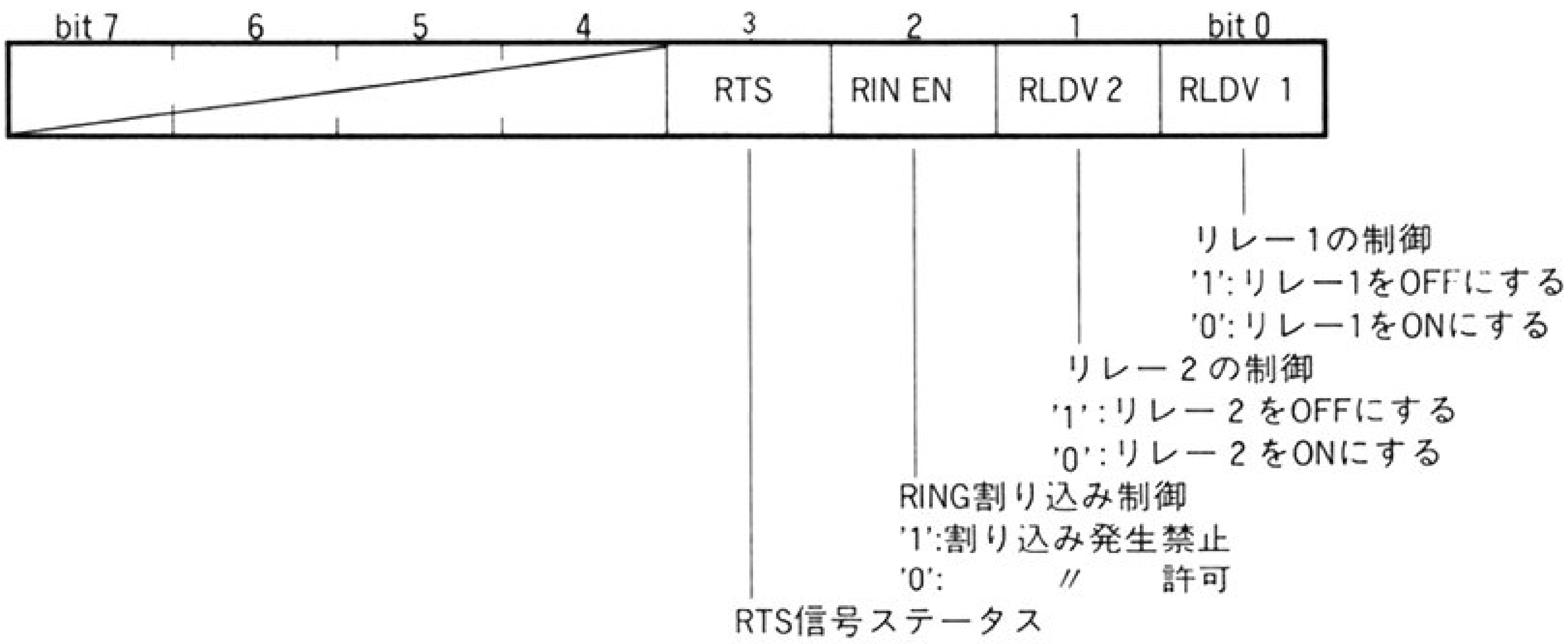
●図……18　8255 ポートA



●図……19　8255 ポートB

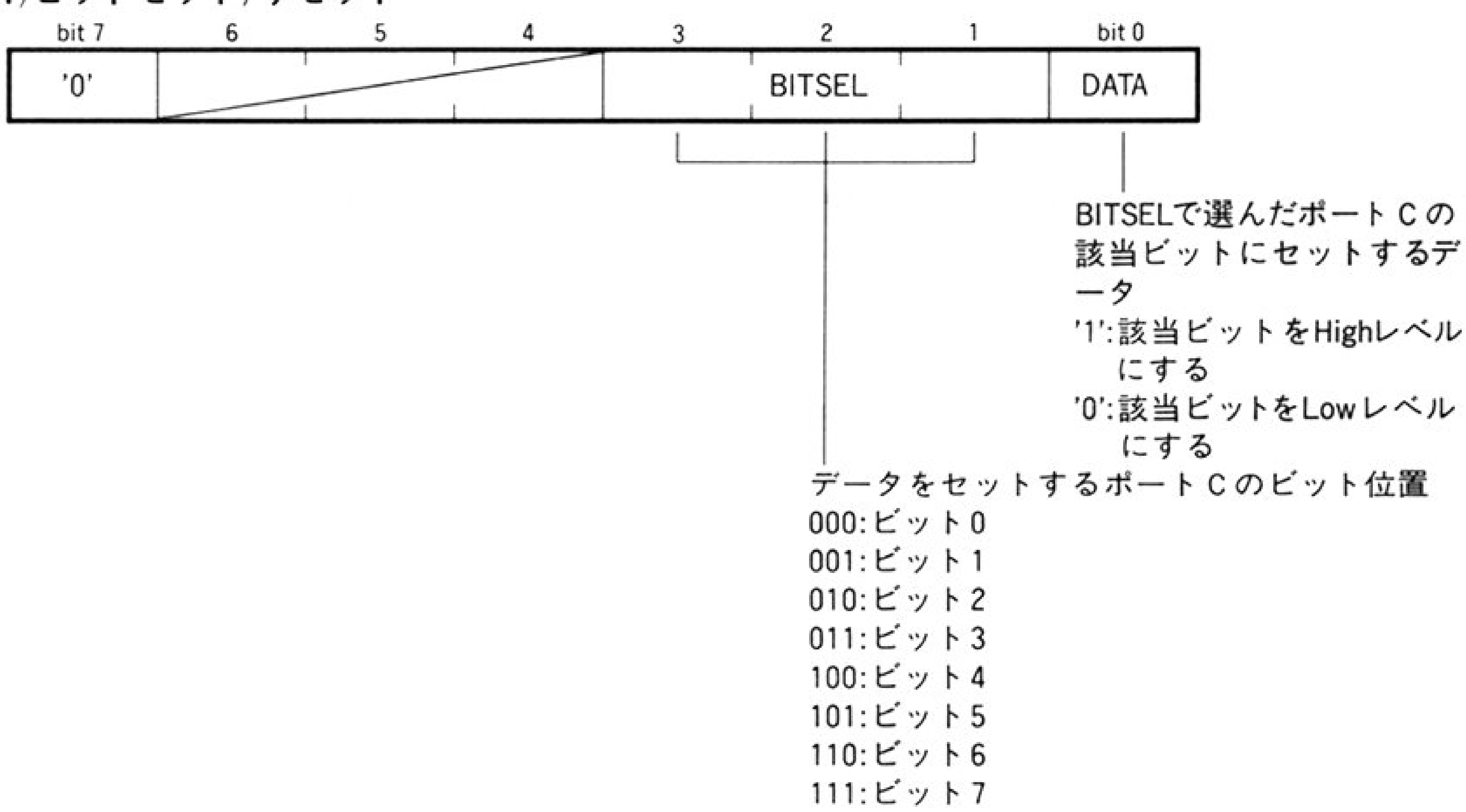


●図……20　8255ポートC



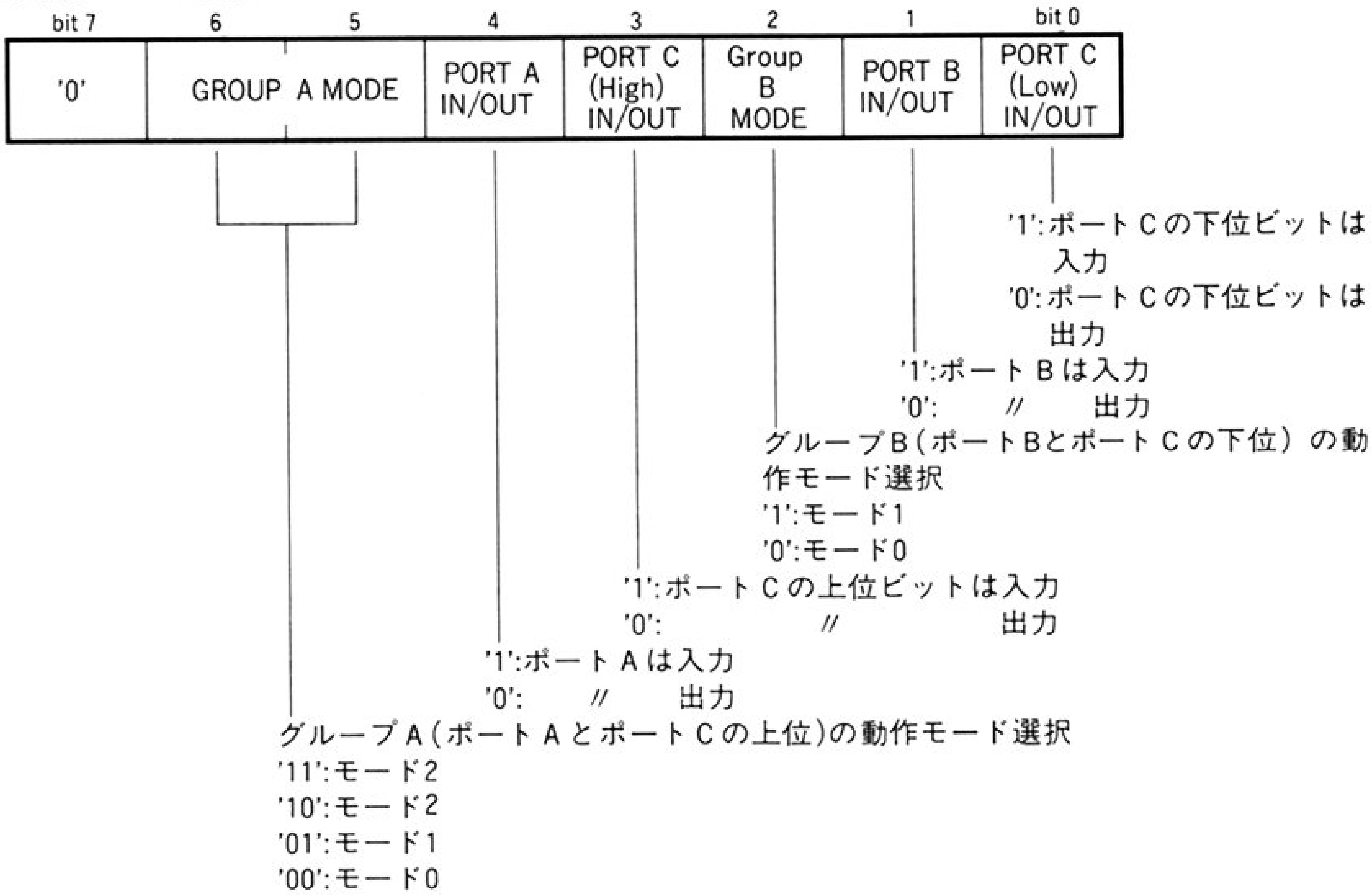
●図……21　8255コントロールワードレジスタ

(1)ビットセット/リセット



●図……21　8255 コントロールワードレジスタ（つづき）

(2)動作モード選択

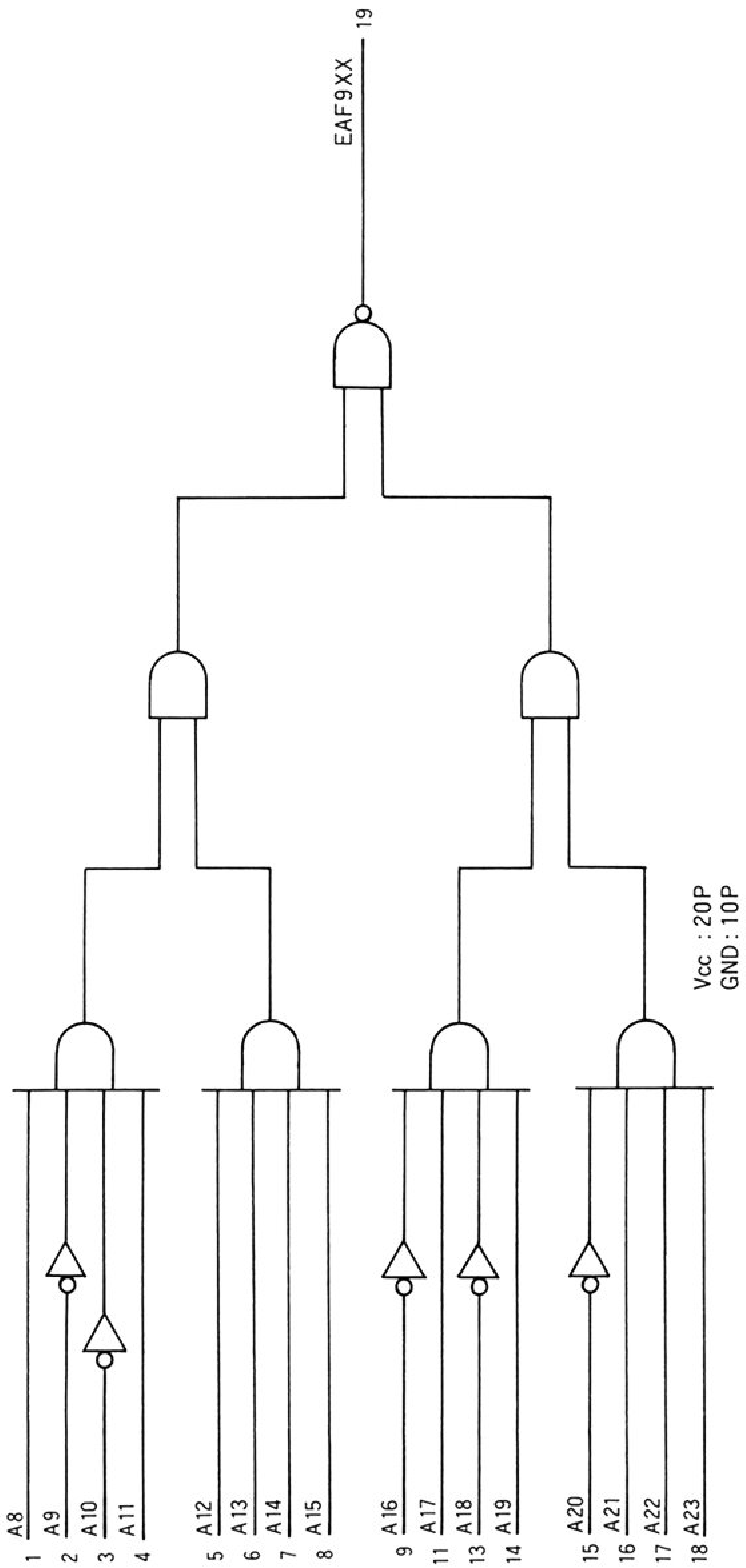


●6　回路図

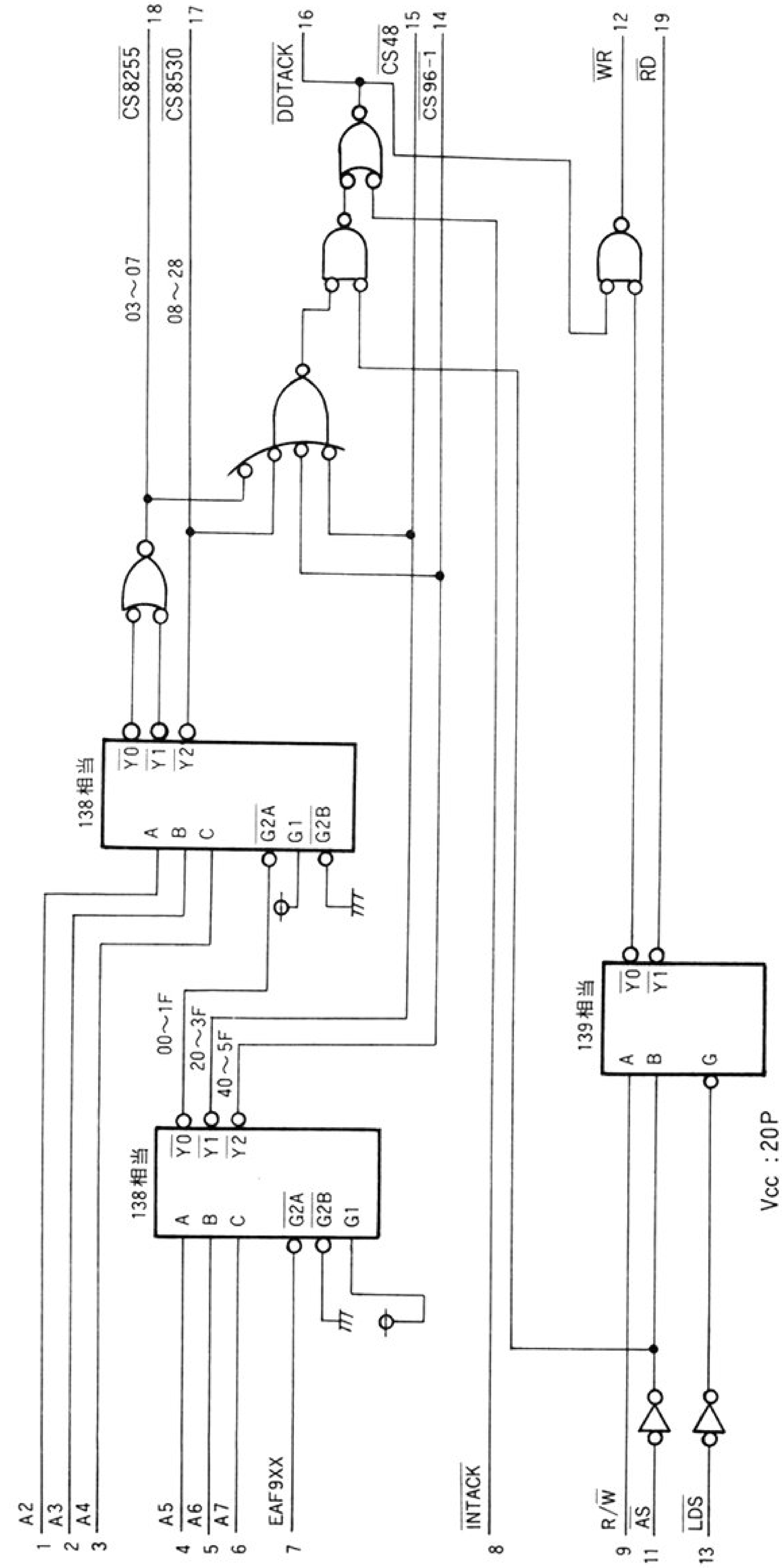
CZ-6BC1の回路図を図22，およびPAL 1とPAL 2の等価回路をそれぞれ図23と図24に示します。

●図……22　CZ-6BC1の回路図（巻末参照）

●図……23　PAL 1 の等価回路



●図……24　PAL 2 の等価回路



ユニバーサル I/Oボード (CZ-6BU1)

単にLEDやスイッチなどをつなぐだけでなく，入出力機能ももたせたユニバーサルI/Oボードは，パラレルボードの相互通信の機能に注目し，機能向上を図ったものとみることもできるでしょう。

ユニバーサル I/O ボードは入出力各 16 ビットずつのデジタル入出力を行うボードです。データ入出力のほかに 8 本のハンドシェーク用信号，1 本の割り込み専用入力を持っており，各種の周辺装置とのインタフェースとして利用することができるようになっています。

データ信号の入出力バッファ（4 個）はいずれもソケット付きとなっており，IC の交換によって入出力ポートの正論理/負論理の変更が可能となっています。

1 仕様

データ信号

入出力論理（出荷時）	負論理（バッファ IC 交換により正論理にも変更可能）
入力データ点数	16 ビット 8 ビット/16 ビット単位で入力可 入力ストローブによるラッチ/割り込み発生可能
出力データ点数	16 ビット

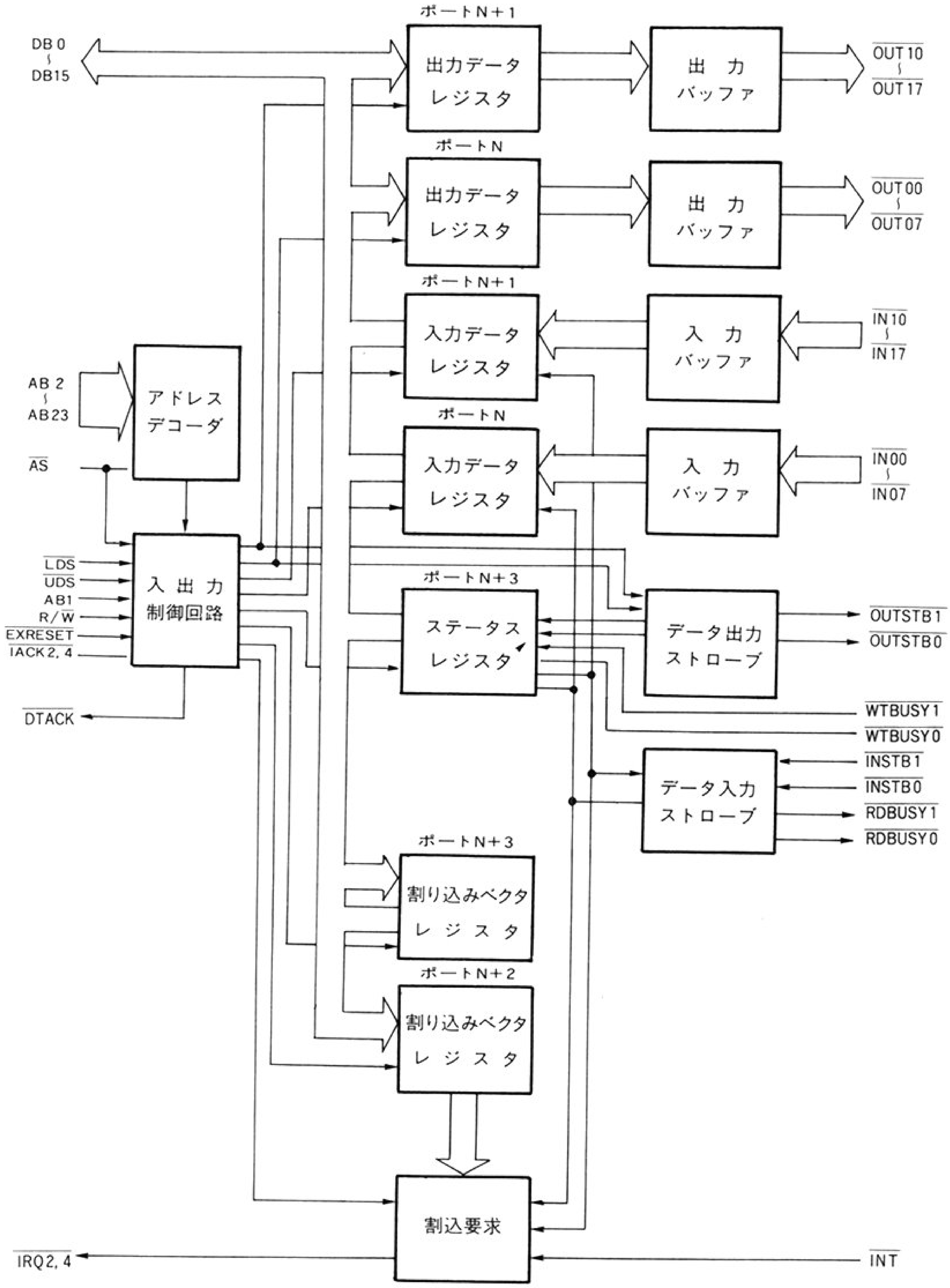
	8ビット/16ビット単位で入力可
	出力ラッチ付き
外部インタフェース	TTLレベル
	全入出力信号にプルアップ・プルダウン抵抗付き
ハンドシェーク信号	
入力ストローブ	2点（データ8ビットごとに1点）
出力ストローブ	2点（　　〃　　）
リードビジー	2点（　　〃　　）
ライトビジー	2点（　　〃　　）
割り込み入力	1点
I/Oコネクタ	50P×2フラットケーブルコネクタ（OMRON XG4A-5039-A）
電源	
電源電圧	＋5V
消費電流	480mA（MAX）
ポートアドレス	$EAFD00～$EAFDFFのうち連続した4バイト
	（先頭アドレスの下位2ビットは常に'00'）
割り込み	レベル2/レベル4（ジャンパースイッチで選択）

●2 回路概要

CZ-6BU1のブロック図を図1に示します。CZ-6BU1は8255などのI/O用LSIは使用しておらず，すべて汎用TTLの組み合わせで作られています。

16ビットずつの入出力のほかに，CZ-6BU1から外部への転送ハンドシェーク信号が2組（4本），外部からCZ-6BU1への転送ハンドシェーク信号が2組（4本），割り込み専用入力信号が1本用意されています。16ビットの入出力ポートはそれぞれ8ビットずつに分けて使用することができるようになっており，ハンドシェーク信号もそれに対応して2組用意されています。

●図……1　CZ-6BU1のブロック図



出力ハンドシェーク信号のうちOUTSTBは，X68000が外部機器に対して，データライン上に有効なデータが乗ったことを示すパルス信号で，外部機器はこれに対してWTBUSYで応答します。WTBUSYは相手機器がデータを受け取るとアクティブになり，取り込みが終了するとインアクティブになります。OUTSTBパルスを16ビットの入出力ポートのうち，どの8ビットポートのいずれをアクセスしたときに出力するようにするかは自由に選択できるようになっています。

入力ハンドシェーク信号のうち，INSTBは外部機器がX68000に対してデータライン上にデータがセットされていることを示すものです。これがアクティブ状態からインアクティブに変化した（仮に立ち上がりエッジと呼ぶことにします）を検出してCZ-6BU1はRDBUSY信号で応答します。RDBUSY信号は，X68000がデータを取り込んだか否かを外部機器に対して示すもので，INSTBの立ち上がりエッジを検出すると自動的にアクティブ（ビジー状態）となり，X68000がデータポートを読み込むと自動的にインアクティブになります。

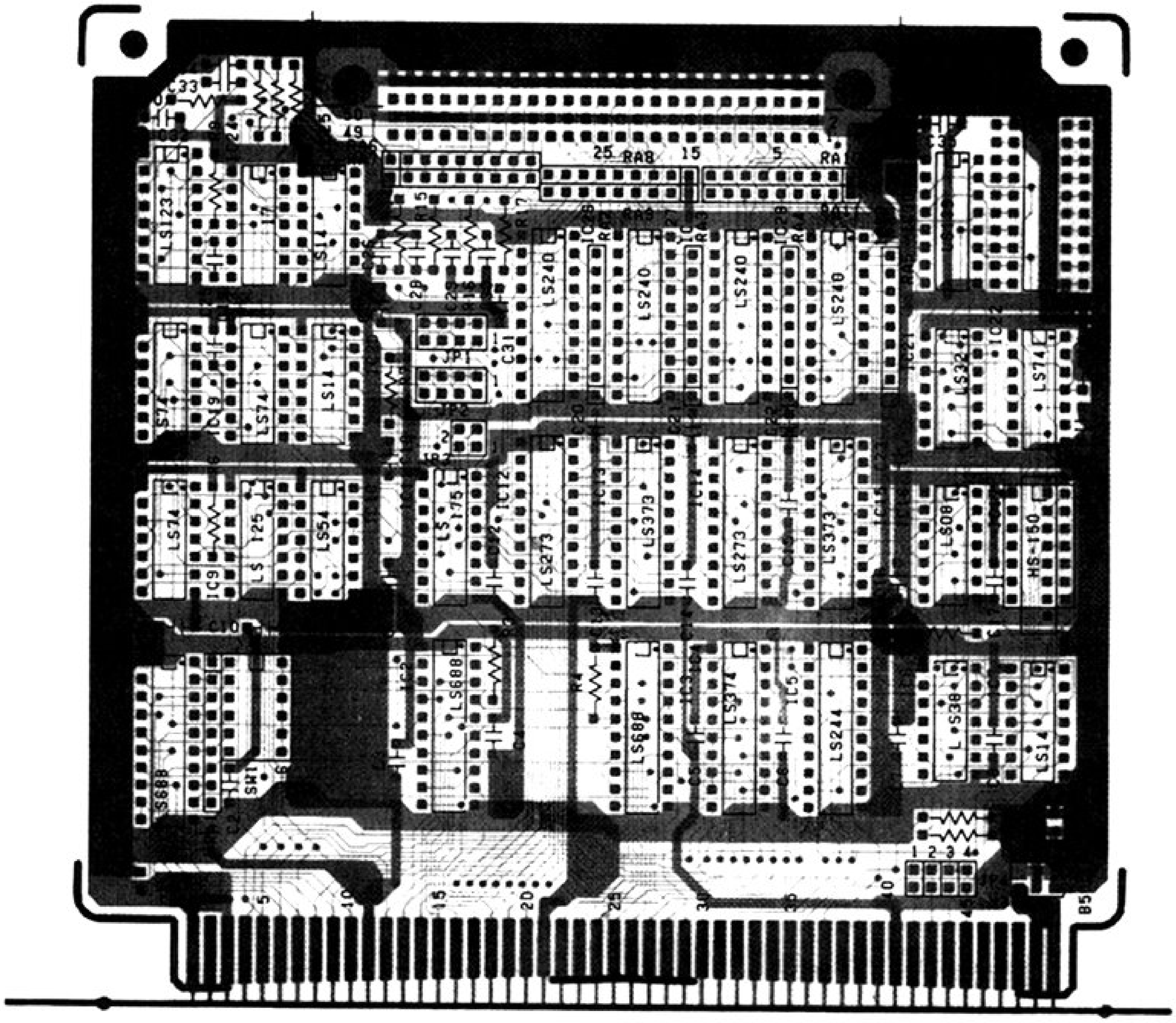
入力ポートはX68000本体が読み出した時点の外部状態を読むようにするか，INSTBの立ち上がりエッジ発生時のデータを保持するようにするかを選択できるようになっています。

割り込みは，専用入力端子によるほか，INSTBの立ち上がりエッジ検出時に発生させることもできるようになっています。

●3 主要部品配置

CZ-6BU1の部品配置を図2に示します。

●図……2　CZ-6BU1の部品配置



●4 設　定

❹·1 ポートアドレスの設定

CZ-6BU1のポートアドレスはディップスイッチ(SW 1)で設定できるようになっています。SW 1の1～6の各スイッチがそれぞれA 2～A 7 (アドレスの2～7ビット目) に対応しており，ベースアドレスを$EAFD00～$EAFDFCのどこにするかを設定します。

スイッチがONのとき該当するアドレスビットが'0'，OFFのときは'1'とすることを示します。たとえば，SW1の1，2，3がOFF，4，5，6がONなら，ベースアドレスの下位

8ビットは'00011100'ですから$EAFD1Cとなりますし，SW1の1，2，3がONで，4，5，6がOFFなら下位8ビットは'11100000'ですから$EAFDE0となります。

❹·2 出力ストローブ信号(OUTSB1/2)の発生条件設定

それぞれ16ビットずつ用意された入出力ポートのうち，どの8ビットポートをアクセスしたときにOUTSB 1/OUTSB 2信号のどちらが出力されるのかを，それぞれJP 1とJP 2で設定します。

4つあるジャンパーピンのうち，1をショートすると出力ポートの下位ビット（ベースアドレス＋1番地），2は出力ポートの上位ビット（ベースアドレスと同一番地）への書き込みで，3，4はそれぞれ入力ポートの下位ビット，上位ビットの読み出しでOUTSB信号を発生するようにするものです。

工場出荷時にはJP 1は1に，JP 2は2にショートピンが入っています。2つ以上の条件でOUTSB信号を発生するようにはできませんので注意してください。

❹·3 データラッチの選択

CZ-6BU1はINSTB 1の立ち上がりエッジでデータの下位側を，INSTB 2の立ち上がりで上位側のデータをラッチ（保持）する機能がありますが，この機能を使うか否かをJP 3で選択できるようになっています。

JP 3の1側をショートすると，INSTB 1の立ち上がりでのデータラッチを，2側をショートすると，INSTB 2の立ち上がりでのデータラッチを行うようになり，オープンにすると本体がデータポートを読み出した時点のデータ入力ラインの状態がそのまま読めるようになります。

ラッチ機能を使った場合，一度データがラッチされると本体側がデータを読み出すまで新たなINSTBの立ち上がりエッジがあってもデータの更新は行われませんので注意してください。

❹·4 割り込みレベルの設定

CZ-6BU1は割り込み専用入力やINSTB信号の立ち上がりエッジで割り込みを発生させることができるようになっています。この割り込みにレベル2とレベル4のいずれを使用するかを選択するのがJP 4です。

レベル2を使用するときはJP 4の1と3を，レベル4を使用するときはJP 4の2と4をショートします。

●5 I/Oマップ

CZ-6BU1のI/Oマップを図3に，各ビットの内容を図4～8に示します。ベースアドレス+0と+1がデータ入出力ポート，ベースアドレス+2が割り込みマスク，ベースアドレス+3が割り込みベクタの設定とハンドシェーク信号ステータス（WTBUSYとINSTB）の読み出しポートとして使用されています。

●図……3　ユニバーサルI/OボードのI/Oアドレスマップ

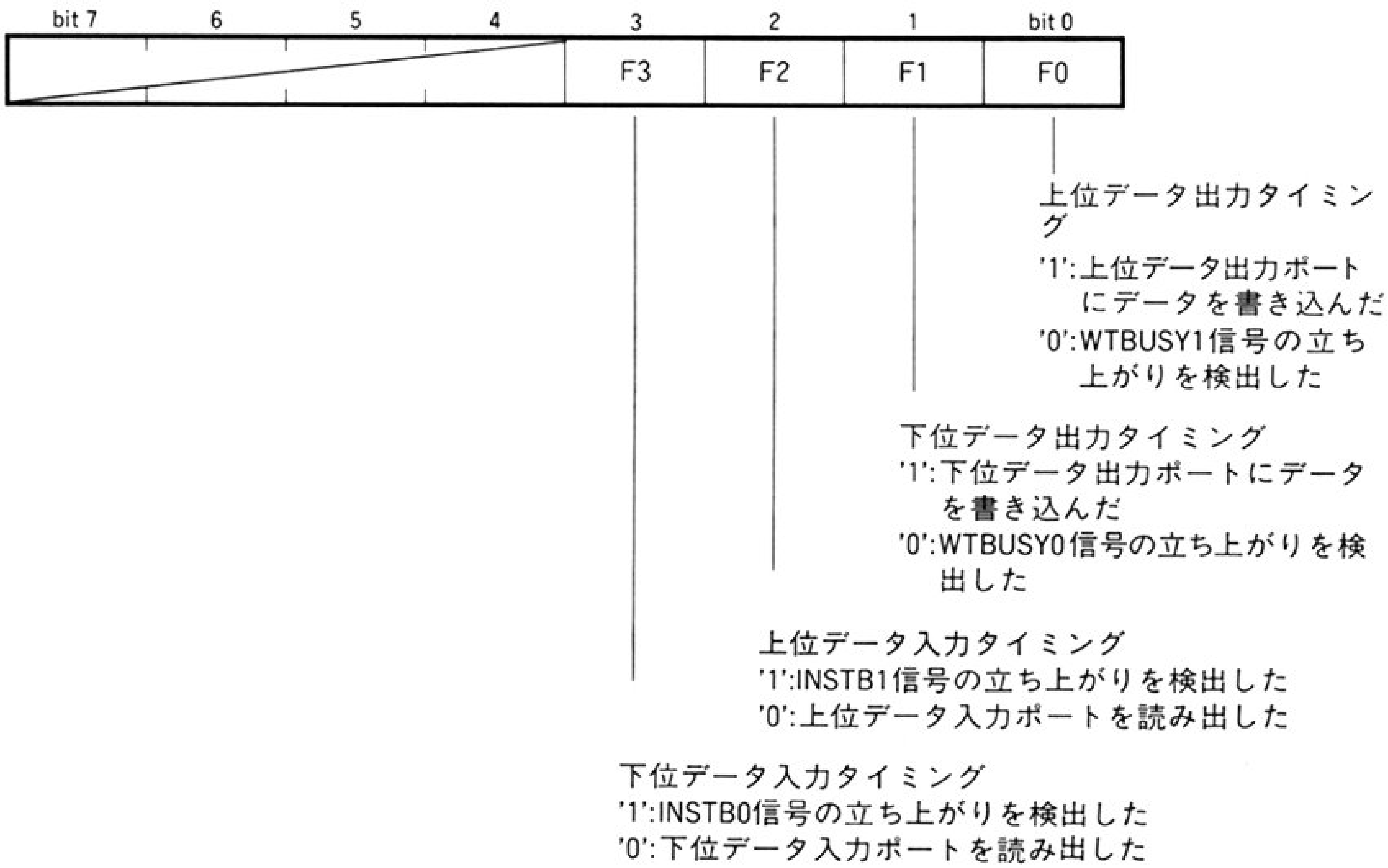
アクセス	アドレス	レジスタ	7	6	5	4	3	2	1	0
Read	ベースアドレス＋$00	上位データ入力	Input Data (High)							
	＋$01	下位データ入力	Input Data (Low)							
	＋$02	未使用								
	＋$03	ステータス入力					F3	F2	F1	F0
Write	ベースアドレス＋$00	上位データ出力	Output Data (High)							
	＋$01	下位データ出力	Output Data (Low)							
	＋$02	割り込みマスク						M2	M1	M0
	＋$03	割り込みベクタ設定	Interrupt Vector							

ベースアドレス：$EAFD00～$EAFDFCのうち，下位2ビットが'00'である値
（$EAFDX0, $EAFDX4, $EAFDX8, $EAFDXC）

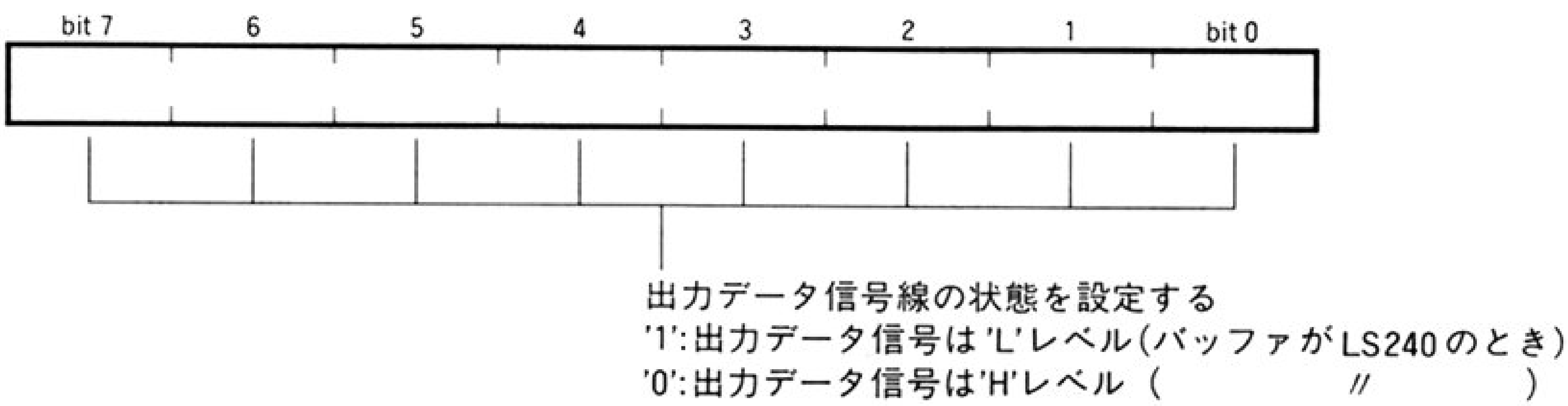
●図……4　上位データ入力/下位データ入力（ベースアドレス+0/+1）



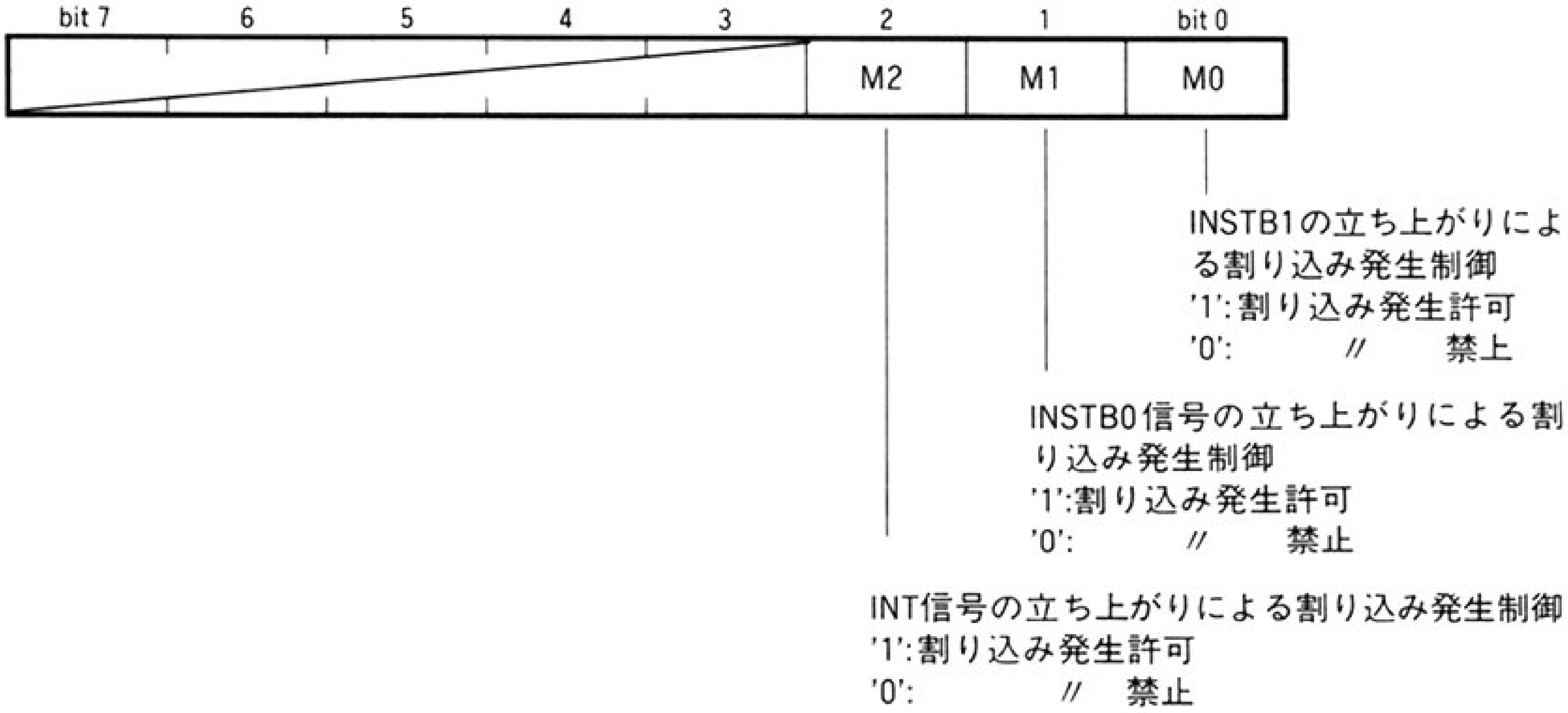
●図……5　ステータス入力（ベースアドレス+3）



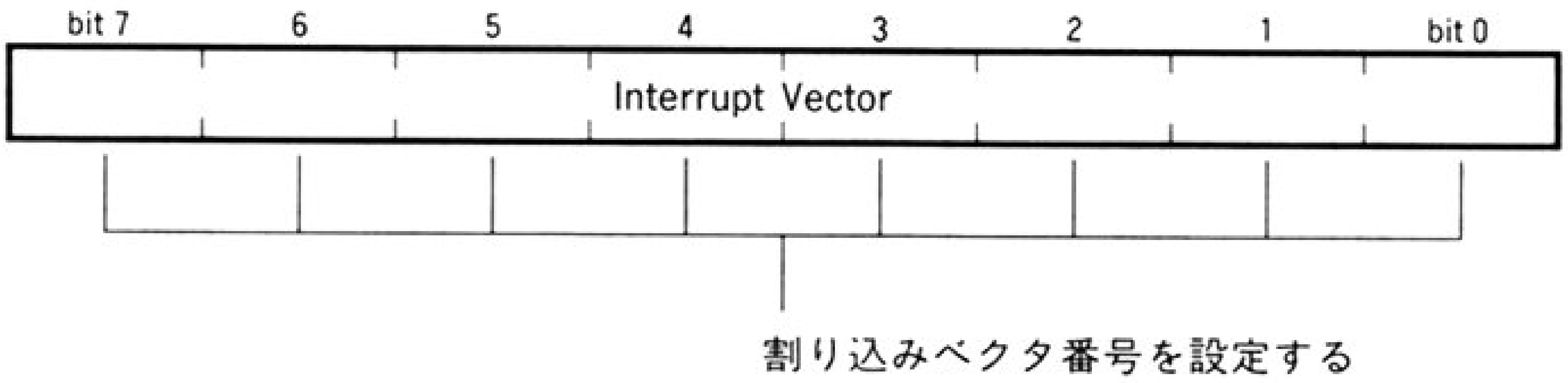
●図……6　上位データ出力/下位データ出力（ベースアドレス+0/+1）



●図……7　割り込みマスクレジスタ（＋2）



●図……8　割り込みベクタ設定（ベースアドレス＋3）



6　コネクタ信号配置

CZ-6BU1 のコネクタ信号配置を図 9 に示します。

●図……9　ユニバーサルI/Oボードのコネクタ信号配置

CN1

1	OUT10	2	GND
3	OUT11	4	GND
5	OUT12	6	GND
7	OUT13	8	GND
9	OUT14	10	GND
11	OUT15	12	GND
13	OUT16	14	GND
15	OUT17	16	GND
17	OUT00	18	GND
19	OUT01	20	GND
21	OUT02	22	GND
23	OUT03	24	GND
25	OUT04	26	GND
27	OUT05	28	GND
29	OUT06	30	GND
31	OUT07	32	GND
33	OUTSTB1	34	GND
35	OUTSTB0	36	GND
37	WTBUSY1	38	GND
39	WTBUSY0	40	GND
41		42	GND
43		44	GND
45		46	GND
47		48	GND
49		50	GND

CM2

1	IN10	2	GND
3	IN11	4	GND
5	IN12	6	GND
7	IN13	8	GND
9	IN14	10	GND
11	IN15	12	GND
13	IN16	14	GND
15	IN17	16	GND
17	IN00	18	GND
19	IN01	20	GND
21	IN02	22	GND
23	IN03	24	GND
25	IN04	26	GND
27	IN05	28	GND
29	IN06	30	GND
31	IN07	32	GND
33	INSTB1	34	GND
35	INSTB0	36	GND
37	RDBUSY1	38	GND
39	RDBUSY0	40	GND
41	INT	42	GND
43		44	GND
45		46	GND
47		48	GND
49		50	GND

●7　回路図

CZ-6BU1の回路図を図10に示します。

●図……10　CZ-6BU1の回路図（巻末参照）

自作周辺機器編

自作周辺機器例

コンピュータに自作の周辺機器を接続して自分の思い通りに動かしたり，これまでできなかったことができるようになるのを見るのは楽しいものです。この編ではX68000のI/Oポートを利用したちょっと変わった周辺装置を紹介します。

コンピュータに自作の周辺機器を接続して，それまでなかったような機能を追加するというのは，非常に魅力のあることです。市販されている製品ではちょっと物足りないところを補ってみたり，ごく単純な機能だけに絞り込んで安く作ってみたりと，自分の好きな仕様にすることができるというのもおもしろいところです。ただ，最初からあまり複雑なものに挑戦しても部品集めなど，作る前の段階や，製作が難しすぎて途中で放り出してしまうようなことになりかねません。

この編では私が実際に製作した周辺機器のうち，比較的簡単に製作できそうなものを選んで紹介することにします。本体との接続はいずれもジョイスティックポートなど，本体に標準で用意されているインタフェースを利用していますので，これらのポートの利用方法の参考としても使えることと思います。ここで紹介する周辺機器は次のとおりです。

・乱数発生機
・ラジコンスティック
・赤外線万能リモコン
・CRT切り替え機

2番目のラジコンスティックはほかのものに比べるとやや複雑ですので，少し製作に慣れた後で挑戦したほうがよいかもしれません。

乱数発生機

純粋に確率でしか論じることができない物理現象を利用して作り出す物理乱数は，物理的な挙動のシミュレーションなどには欠かすことができないものです。ここでは，小型版の物理乱数の発生装置を自作してみることにします。

サイコロや電気ノイズのような物理現象を利用した乱数を物理乱数といいます。これに対してパソコンなどで一般的に用いられている，計算によって作り出す一見デタラメな数字の並びを疑似乱数といいます。疑似乱数の発生方法はいろいろなものが考案されていますが，そのなかでもよく使われているのが，割り算をしたときの余りを使う，合同法と呼ばれるやり方です。ある整数Xがあったとき，ａＸ＋ｂをｃで割った余りＹを次の乱数値とし，次はａＹ＋ｂをｃで割った余りを計算するといったぐあいに計算していくわけです。ａ，ｂ，ｃの選び方をうまくやると周期が長く，わりと質のよい乱数が得られます。

疑似乱数は計算で発生させているため，当然，再現性があります．初期条件を同一にすれば何度でも同じ数の並びが生成されます．これは目的によっては便利なのですが，乱数の質が問題となるような用途には使用できません。ある程度大きなコンピュータでは乱数発生機というハードウェアが用意されていて，物理乱数を得られるようにしていますが，パソコンで乱数ボードなるものが売られているのを見たことはありません。雑誌でも取り上げられたことはほとんどないようで，少なくとも私の手元の本のどれにも物理乱数の発生などというものは取り上げられていませんでした。

●1 乱数発生の実験

何をもとに乱数を作るかということが最初に問題になります。比較的大きなノイズを発生するデバイスとしてよく知られているのはツェナーダイオードです。くだんの大型計算機の乱数発生機もツェナーダイオードを利用しているとのことでしたが，実験してみると思ったほど大きなノイズは出てきません。もう少し扱いやすいものがないかと探していたところ，トランジスタに逆電圧をかけたときの降伏を利用するとよさそうであることがわかりました。ノイズというより発振に近い状態のようですが，シンクロで見た限りでは周期性はほとんどないようですし，なにより出力がトランジスタの1段増幅だけで電圧レベルがV単位まで引き上げられるという，非常に大きく扱いやすい信号だというのが魅力です。再現性もさほど問題なさそうですので，ここではこの信号を利用して乱数発生機を作ってみることにします。

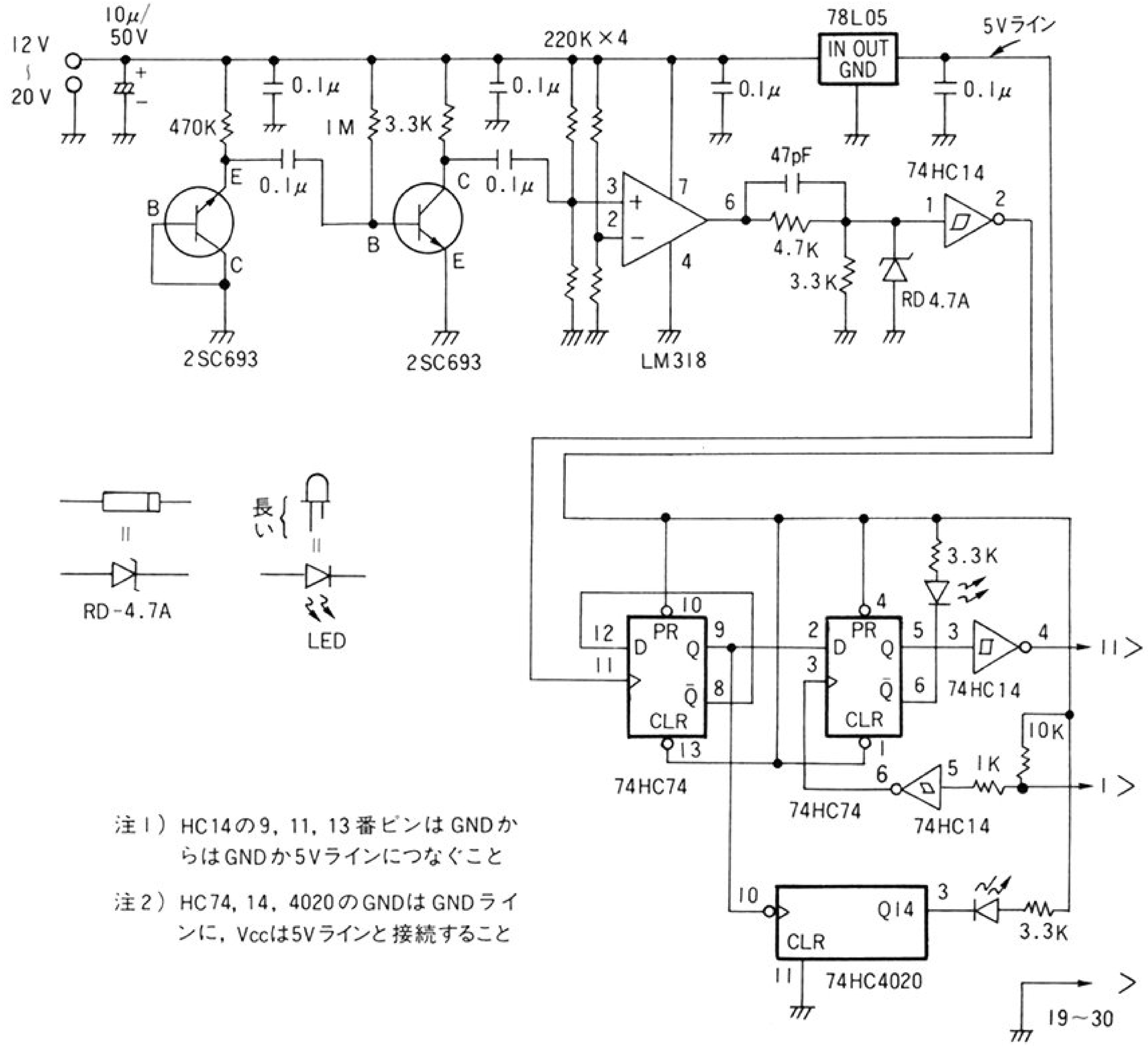
●2 回路設計

なかなか慣れないものであったこともあり，何度かカット&トライを繰り返してできた回路が図1です。この回路について，簡単に説明を加えておきましょう。

❷·1 電源について

乱数発生機は，各種のパソコンに接続できるよう，プリンタインタフェースを使用することにしました。トランジスタの降伏点は10V程度なので，9Vの電池(006P)を2個直列にして使うことにしました。ニッカド電池やリチウム電池と違って一般の乾電池の電圧は新品のときからだらだらと電圧が落ちていくような特性を持っているので，20Vから12Vくらいまで電圧が変化しても安定して動くことを確認しておきました。

●図……1　回路図



❷·2　雑音源部分

回路図の端にある，トランジスタを奇妙な向きに接続している部分が雑音発生源です。トランジスタのコレクタとベースを直結してGND(0 V)に落とし，エミッタは抵抗を通して電源に接続され，ほんの少しだけ逆方向に電流が流れるようにしておきます。470 Kとやや大きめの値にしているのは，電流が増えるにつれて雑音レベルが下がっていく現象が見られたからです。

さすがに電源電圧が2倍近く変化すると，流れる電流も雑音の発生ぐあいも変化していきます。試しにノイズ発生用のトランジスタに流す電流を徐々に増していったところ，電流が増すに従って次第にノイズ出力が減っていき，最後にはごく小さな波形になってしまいました。雑音源として使うには，あまり電流を流さないようにしなくてはならないようであることがわか

ったので，大きめの抵抗にして 20 V以上でも安定して雑音が発生することを確認しておきました。

❷·3 信号増幅部

発生した雑音を次段のトランジスタで増幅します。間にある 0.1 μF のコンデンサは，10 V 前後の直流成分をカットし，交流成分である雑音信号だけを次段のトランジスタに与えるためのものです。バイアスはいちばん簡単な固定バイアスで，コレクタ電流は 1 mA くらい流しておくことにします。私が大量に買い込んだトランジスタは 2SC693E と最後に E がついたものだったのですが，一応，電源が 12 V で hFE を 80 とコレクタ電流がいちばん少なくなる条件で 1 mA くらい流れるように計算します。hFE を 80 とし，コレクタ電流が 1 mA ならベース電流は 1/80 mA です。ベースの電圧は，ほぼ 0.6 V と固定ですから，残りの 12−0.6=11.4 V が抵抗にかかる電圧です。後はオームの法則で，

$$R=E/I=11.4/(1/80)=912(K\Omega)$$

ここから，抵抗の値は 1 M(=1,000 K)Ω くらいでよかろうということになりました。ちなみに，コレクタ電流が最大になるのは電圧が 20 V(一応 20 V を上限と考えておきます)，hFE が最大（160）のときです。このときのベース電流は，

$$I=E/R=(20-0.6)/1000=0.0194(mA)$$

コレクタ電流は，この hFE 倍ですから 0.0194 mA×160=3.10 mA となります。

これだけ動くと，コレクタの負荷抵抗をどのくらいにすべきか，少々考えさせられます。教科書的に考えていくなら，IC-VCE 特性図に負荷線を引いて考えるのでしょうが，そこまで真剣に考えるほどの増幅回路ではないので，コレクタ電圧が 5 ～ 6 V以上ならよいという方針で簡略設計で間に合わせます。電源電圧が低く，12 V，hFE が 160 のときにコレクタ電流は，

$$(20-0.6)/1000\times 160=1.8(mA)$$

くらいです。ここで，コレクタ電圧が 6 V とすると，抵抗値は，

$$R=E/I=6/1.8=3.3(K\Omega)$$

となります。

hFE が小さく，コレクタ電流が 1 mA のとき，この抵抗の両端の電圧は 3.3 V と少々小さく，入力の雑音が大きいときには出力波形の頭がクリップする可能性もありますが，別にオーディ

オアンプにしようというのではありませんからかまわないでしょう。

❷・4 波形整形部

増幅段で大きくなった雑音は，やはりコンデンサを通って次段のオペアンプに入力され，デジタル的な波形に整形されます。オペアンプをこのように使うと，コンパレータ(電圧比較器)として動作します。＋入力と－入力の電圧が高ければ，出力は＋電源電圧くらい(かりに1としておきます)，－入力のほうが高ければ－電源電圧付近の電圧(かりに0としておきましょう)になります。つまり，どちらの電圧が高いかということによって2値信号（デジタル信号）に変換できることになります。今回はどちらの入力も220Kの抵抗で，電源を1対1に分割し，片方にいましがたトランジスタで増幅した雑音を放り込みます。通常，入力は抵抗の誤差くらいの電圧差しかありませんから，出力はぎりぎりのところでどちらかに転んでいる状態です。ここで，片方が雑音によって振り回されるために，出力が1になったり0になったりするわけです。

オペアンプにはLM318を使用しました。オリジナルはナショナルセミコンダクタですが，フェアチャイルド，テキサス，日電，AMDなどもセカンドソースを作っているので入手は容易であると思います。LM 318は帯域幅が15 MHz，スルーレート（出力の変化する速度）が50 V/μSとなかなか高帯域で高速なオペアンプで，今回のように周波数が高めのデジタル信号（方形波）を得ようというのには好都合です。

❷・5 信号レベル変換方法の検討

オペアンプの出力は方形波信号であり，デジタル信号風であるとはいえ，電圧レベルは電源電圧近くまでとかなり高いので，このまま5Vという低い電圧の世界で動いているデジタルICに接続するわけにはいきません。また，一般にオペアンプの出力は電源電圧よりも2Vから3V程度の電位差があるのが普通です。オフセットを3Vと考えれば，0のときには3V，1のときは電源電圧が12Vなら9Vの出力となります。

コンピュータとつなぐには，これをデジタルICの論理電圧レベルに変換しなくてはならないのです。電源電圧が2倍近くも変わることから電源電圧が12VのときになんとかHレベルになるようにしておいて，高くなりすぎる分はツェナーダイオードを使ってリミットをかければなんとかなります。むしろ問題はLレベルです。出力が3VというのはC-MOSのICを使うにしても高すぎます。

いろいろ考えたのですが，Ｌレベルをきれいに０Ｖ近くまで落とすことはあきらめて，出力を適当に抵抗で分圧することで，デジタルICがＬレベルと認識できるくらいの電圧にしてしまうことにしました。当然，分割比をきちんと計算しておかないと信号が全部ＬレベルやＨレベルになってしまいますので少々気を使います。

❷·6 レベル変換回路

オペアンプの出力には4.7 Vのツェナーとコンデンサを入れました。オペアンプの出力が20 Vまで上がったとしてツェナーの消費電力を計算しておきます。まず4.7 Kの抵抗に流れる電流は，

$$(20-4.7)/4.7=3.26\,(\mathrm{mA})$$

です。このうち，3.3 Kの抵抗に流れる分は4.7/3.3＝1.42(mA)ですから，残りの3.26−1.42＝1.84 mAがツェナーに流れる電流です。したがって，ツェナーの消費電力は1.84×4.7＝8.65(mW)となります。いまどきのツェナーは小さいものでも200 mWくらいですから，十分でしょう。この余裕に期待して追加したのが4.7 Kに並列につないである47 pFのコンデンサです。

抵抗で分割して取り出した波形は，教科書どおりの世界であれば，入力信号と完全に対称になるはずなのですが，方形波が相手の世界ではそれほど単純ではありません。現実には部品の足や基板，そして部品内部でも，いたるところにコンデンサが形成されていますのでツェナーにさらに並列にコンデンサをつないだような状態になっているのです。オペアンプの出力が0になるときは，このたまった電荷を抵抗を通して放電しなくてはなりません。逆に1になるときは抵抗を通して充電してやらなくてはならないわけです。このため，波形の立ち上がりや立ち下がりがなまってきます。これを補償する目的で入れたのが47 pFのコンデンサです。立ち上がりや立ち下がりのときには抵抗だけでなく，このコンデンサを通して一気に電荷の充放電を行うわけです。

このコンデンサの容量が少ないと，波形のなまりが残りますし，容量が大きすぎると，立ち上がりや立ち下がりがいったん行き過ぎた後，今度は跳ね返るようにしばらく暴れる現象（オーバーシュート/アンダーシュート）が出やすくなりますので，注意して選択する必要があります。

計算するより実験してしまったほうが簡単そうなので波形を観測しながらカット＆トライで決めました。私は手持ちの都合で27 pFにしたのですが，少し容量が足りず，100 pFでは少々補償のやりすぎでしたので，回路図では47 pFとしておきました。もし，シンクロスコープ(ス

トレージオシロがあればベストですが）が使えるようでしたら，波形を見ながら調整するとよいでしょう。

❷·7 データ取り込み部

このようにして，トランジスタで発生した雑音がようやくデジタル ICのレベルにまで変換されました。ただし，このまま読み取ると，1と0の発生する確率がノイズの波形に依存してしまうことになるため，フリップフロップで分周して1と0の発生確率が1対1になるようにします。クロック波形がある程度きれいになるように，シュミット特性を持ったHC 14を通しています。

このフリップフロップの出力は，当然のことながら常時変化しています。これをパソコンに読み取らせる場合，そのままにしておくと，読んでいる途中でデータが変化する可能性が非常に高くなります。安定した読み取りを保証するために，この出力をラッチする回路をつけておきます。つまり，パソコン側から発生したパルスでデータをラッチさせておいて，後からゆっくりと読み出しにいこうというわけです。ラッチしたデータは，次のパルスを出力するまで変化しませんから，気楽に読み出せます。出力は HC 14 でバッファして取り出しました。

「掛け声」に相当する信号も HC 14 を通します。入力に直列に入っている抵抗は ICの電源電圧よりも入力電圧が高いときのための入力保護です。抵抗で5Vとつないでいるのは，入力がオープンになったときに入力がふらふらしてラッチアップを起こさないようにすることと，パソコン側の出力レベルが低くなりがちなので，Hレベルを保証するために入れてあります。74HC4020とLEDは，動作には直接関係のない，おまけ回路です。基板に少々隙間が残っていたので，乱数が発生しているかということと，パソコンの読み込みが行われているかを目で確認できるように入れてみました。

❷·8 使用インタフェース

接続のためのインタフェースとしては，プリンタポートを使ってみました。ビット数が多ければパソコン本体の拡張スロットを利用したほうがよいのでしょうが，今回はたかだか1ビットの乱数ですから，入力端子が1ビット，さらに「掛け声」のための出力端子が1ビットあれば十分です。候補はジョイスティックポートとプリンタポートですが，ほとんどの機種についていて，しかも統一されているので，プリンタポートを使うことにします。BUSYを乱数データ入力，STBをラッチ信号（掛け声）に使っています。

●3 製作上の注意点

精密さを要求される回路ではないので，特に注意しなくてはならないようなところはありませんが，ちょっとしたコツのような部分について触れておきましょう。

❸·1 基板の作成

基板はカッターでパターンを切りながら作っていきます。カッターでパターンを切り離すときにはパターンにキズをつけるだけでなく，適当な幅ではがすようにします。切り口がV字になるようなイメージで，両側から切ると，ペロッとパターンが取れます。もし彫刻刀を持っているなら，V形のものを利用するとよいでしょう。

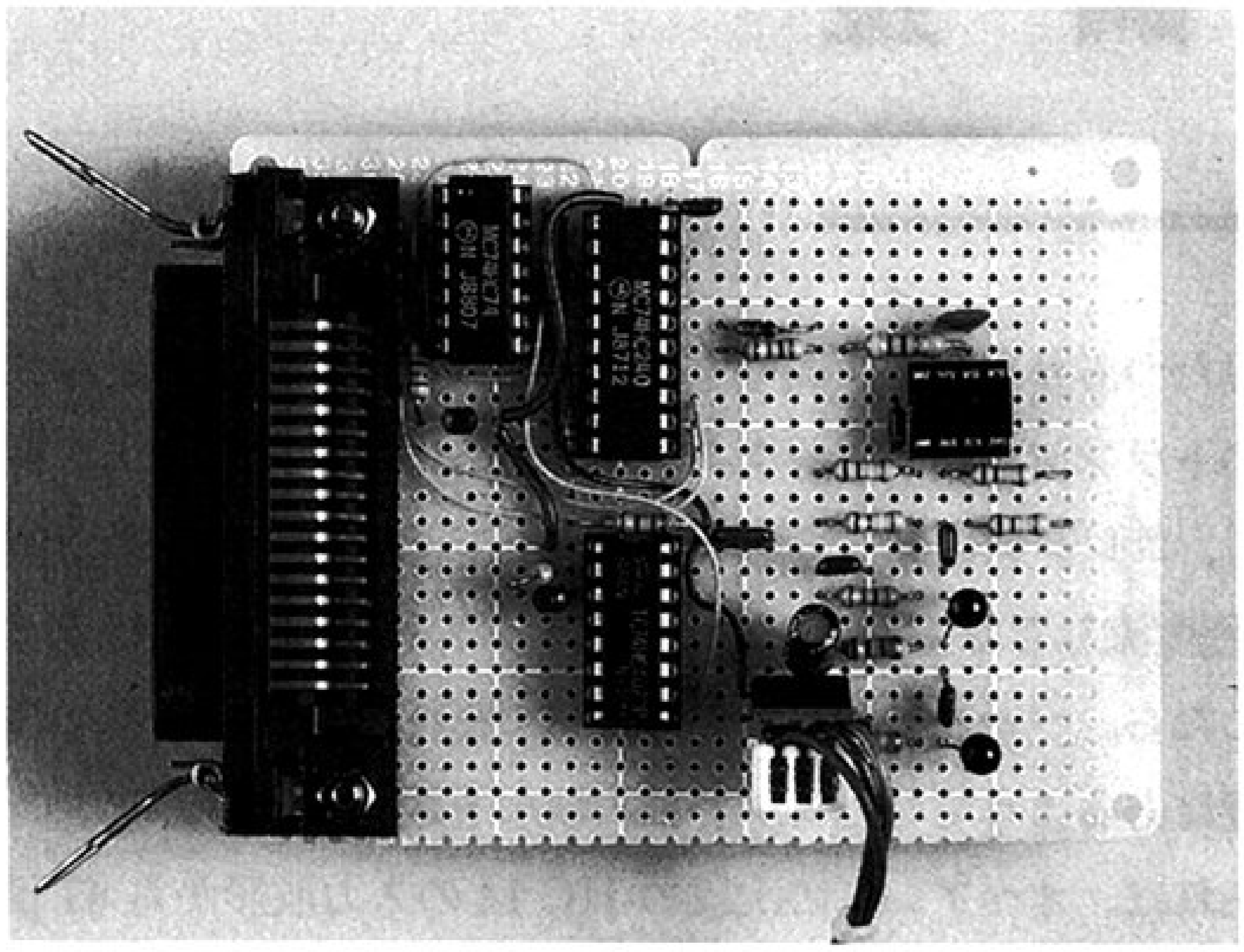
基板を2つに割るときには，表と裏から軽くキズをつけておいてから机のへりなどで曲げれば簡単に割れます。もともと2.54 mmピッチで穴が開いているという状態は，基板にミシン目が入っているようなものですから，無理に力をかければ割れやすくなっているのです。カッターでキズをつけるというのは，この割れやすい状態にくせをつけてやるという作業なのですから，それほど深く切り込む必要はありません。

使用するカッターは一般の紙工作用のもので十分ですが，カットのときにはけっこう力をかけるでしょうから，新規に買うのでしたら，刃をねじで固定する方式のものがよいでしょう。

❸·2 部品配置

部品配置などが決めづらい人は写真を参考にしてください。アナログ部分は，回路図とよく似た配置になっているでしょう。こうしておくと，後々波形を見たり，調整するときにどこを観測しているのかがわかりやすくて楽です。

●写真……1　部品配置



❸・3　電源処理

デジタルICの電源ピンとグラウンドの間には0.1μFのコンデンサを入れておきます。これは電源ラインのノイズを抑えるためのもので，回路図には出てこないので気をつけてください。

❸・4　チェック

完成したら，すぐに電源を入れたりしないで，まずテスターなどで消費電流を測っておきます。ソケットにはICを差し込まずに測ります。もし100 mA以上も流れるようでしたらどこかおかしいところがある証拠ですので，電源を落として配線をチェックしなおしてください。次にICの電源電圧を見ます。デジタルIC(74HCシリーズ)のGNDとVcc端子の間の電圧は5 Vになっているはずです。もし大幅に違うようでしたら，3端子レギュレータまわりなどを見直してください。

OKとなったら，いったん電源を切ってからICを差し込みます。新しいICの足は広がり気味なので，机の上などで真っ直ぐに整形しておきます。足先ではなく，足の根元近くを折り曲げる気持ちでやるときれいにできます。

再度電源を入れてみます。うまく動いていれば，74HC4020につけたLEDがせわしなく点滅を繰り返しているはずです。

●4 調　整

あちこち気をつけたつもりなので，調整する場所はほとんどありませんが，テスターを持っていたら，増幅段のトランジスタのコレクタ電流を測っておきましょう。測定は，パターンを切ったりしなくても，コレクタにつないだ 3.3 K の抵抗の両端の電位差を測って行います。だいたい 1～2 mA 程度流しておく予定ですから，3～7 V 程度になるはずです。あまりにもはずれているようでしたら，ベースについている 1 M の抵抗を調整してください。もしシンクロスコープなどが手近にあるようでしたら，各部の波形を観測しておくとよいでしょう。ストレージオシロなどがあれば，オペアンプの次段の HC 14 の入力波形から 47 pF の調整が容易に行えるでしょう。

さて，物理乱数である以上，発生頻度が問題ですので測定してみました。シンクロ，できたら周波数カウンタで，HC 4020 の 3 番ピン（LED をつけた端子）の周波数を測り，それを 2 の 14 乗倍（16384 倍）すれば，おおよその値は知ることができます。

平均の発生頻度よりも極端に短い周期で取り込むと同じデータが続く場合が多くなりすぎて，とても乱数とは呼べなくなってしまいます。筆者も試作段階を含めて何度か測定してみましたが，だいたい乱数出力（HC 74 の 9 番ピン）の周波数は平均すると 500 kHz から 1 MHz の間であるようです。ランダムな値を得るにはこれよりも十分に長い周期でサンプリングしなくてはなりません。できたら，100 倍，すなわち 100～200 μS くらい間をおきながらサンプリングしたいところですが，1 ビット得るたびに 200 μS，すなわち 0.2 mS かかるというのは用途によっては遅すぎるかもしれません。そのときは実験しながら読み込むインターバルを調整してください。

●5 読み取りソフト

いよいよ X68000 本体と接続します。STB 信号を 0 にして 1 に戻すという操作を繰り返し行うと，そのつど 74HC74 につないだ LED がランダムに点滅するはずです。うまくいっているようでしたら，次に STB にパルスを送った後，BUSY 信号を読み出してみます。1 になっ

たり，0になったりするのが読み出せれば成功です。

6 おわりに

疑似乱数のときのように，乱数の質を心のどこかで気にすることもなく，気分よく使うことができました。汎用をめざしてプリンタポートを使ったために1ビットずつしかデータが得られないことや，周波数がかなり低いため発生速度がかなり遅い点など，改良の余地はまだまだありますが，とりあえず，物理乱数をパソコンで使うという，当初の目的は達成されたと思います。

7 追 伸

ここで使ったトランジスタ，2SC693をもう少し手に入れやすいものと置き換えられないかと調べたところ，2SC1815-Yが使えそうなことがわかりました（表1）。YランクはhFEが100程度であるので抵抗値もそのままで大丈夫です。回路図はそのままで，トランジスタだけを1815-Yにすればよいでしょう。私の手元の基板は差し替えたままになっています。

●表……1　部品一覧

	型　番	数量	単価
トランジスタ	2SC963E（またはF）	2	30
発光ダイオード	TLR102など	2	30
3端子レギュレータ	78L05	1	80
アペアンプ	LM318	1	320
デジタルIC	74HC14	1	50
	74HC4020	1	220
	74HC74	1	100
ツェナーダイオード	RD-4.7A	1	15
電解コンデンサ	10μF/50V	1	25
セラミックコンデンサ	0.1μF	8	15
	47pF	1	15
抵抗	1k	1	5
	3.3k	4	5
	4.7k	1	5
	10k	1	5
	220k	4	5
	470k	1	5
	1M	1	5
コネクタ	アンフェノール36ピン	1	800
	電源供給用	1組	40
電池スナップ	006P用	2	20
乾電池	006P	2	100
プリント基板	ICB-504EG（サンハヤト）	1	300
ICソケット	14ピン	1	15
	16ピン	1	15
	20ピン	1	20
計			2660円

●リスト……1　サンプルプログラム

```
1000 /* 乱数発生機チェック用プログラム
1010 /*  MZ-2500のHu-BASICからの移植です
1020 /*  bpeek,bpoke関数が必要です。電脳倶楽部等で入手したものを利用してく
ださい。
```

```
1030 int a,b,i,l,r,x,y,pi,zi:float f
1040 screen 2,0,1,1
1050 cls
1060 print"0：乱数パターン表示　1：酔歩　2：円周率";:input a
1070 if a=0 then goto 1330
1080 if a=1 then goto 1140
1090 if a=2 then goto 1420
1100 beep:goto 1060
1110 /*
1120 /* 酔歩：どちらに行く率も同じはずです
1130 /*
1140 l=0:r=0
1150 cls
1160 for i=1 to 100
1170   x=39
1180   gosub 1580
1190   locate x,0:print" "
1200   if a=0 then x=x-1 else x=x+1
1210   if x=0  then l=l+1:locate 0,10:print "左  =  ";l;"回";:goto 1250
1220   if x=78 then r=r+1:locate 0,11:print "右  =  ";r;"回";:goto 1250
1230   locate x,0:print"#"
1240   goto 1180
1250 next
1260 locate 0,15:print "ＥＮＤ"
1270 beep
1280 end
1290 /*
1300 /* 乱数のグラフィック表示
1310 /*
1320 /*
1330 screen 2,0,1,1
1340 for y=0 to 199
1350   for x=0 to 639
1360     gosub 1580
1370     if a=1 then pset(x,y,15)
1380   next
1390 next
1400 beep
1410 end
1420 cls
```

```
1430 /*
1440 /* 乱数によって的撃ちを行い、1／4円の中に入る確率から円周率を求めます
1450 /*
1460 pi = 0
1470 for i=1 to 10000
1480    gosub 1670:x = b
1490    gosub 1670:y = b
1500    if x*x+y*y <= 65025 then pi = pi+1
1510    locate 0,0:print using "#.####";pi*4#/i
1520 next
1530 beep
1540 end
1550 /*
1560 /* 乱数発生機からのデータを読み出し、変数ａに入れて返します
1570 /*
1580 /* f = rnd():if f=0.5# then goto 1580
1590 /* if f<0.5# then a=0 else a=1
1600 /* return
1610 bpoke(&HE8C003,0):bpoke(&HE8C003,1)
1620 a = (bpeek(&HE9C001)/32) and 1
1630 return
1640 /*
1650 /* ８ビットの乱数を作成します。価はｂに入れて返します
1660 /*
1670 b = 0
1680 for zi = 1 to 8
1690   gosub 1580
1700   b = b*2+a
1710 next
1720 return
```

ラジコンスティック

サイバースティックよりもはるかに長いアナログスティックとしての歴史を持つ，ラジコンのプロポをサイバースティックにみせかけるインタフェース回路を作成してみました。スティックの扱いやすさはさすがに歴史を感じさせてくれます。

初めての本格的なアナログジョイスティックであるサイバースティックの登場は，ゲームの世界を一新させたといってもよいでしょう。標準的なジョイスティックポートがあれば接続することができるということから，X68000以外の機種のゲームでも“アナログジョイスティックが必要です”というものが売られているようです。

サイバースティックのもうひとつの功績は，これまでのジョイスティックポートになんら特別な細工をすることなく，4チャンネルのアナログ情報と14ビットのデジタル（スイッチ）情報を渡せる方法を提唱したことです（仕様書ではデジタルスイッチは10ビットですが，実際には14ビット分送っています）。現在発売されているサイバースティックでコントロールできるのはアナログ3チャンネルとデジタル10ビットですから，まだまだ拡張性があることになります。

このデータ転送方式を利用して，模型の世界のアナログスティックともいえる，ラジコンのプロポをX68000と接続してみることにしました。もちろん接続といっても，プロポを改造するわけではありません。プロポの出す電波を受け，そこから得たスティックの傾きのデータをサイバースティックと同じ方法で渡してやろうというものです。名付けてラジコンスティック。使ったプロポが2チャンネルのものですのでスロットルの操作はできませんが，スイッチのほうはサイバースティックにあるものはすべて使えるようにし，ついでにフットスイッチもつけてみました。

●1　プロポの選択

まず，いちばん肝心な情報であるプロポがどのようにしてスティックの傾きを伝えているのかを知らなくてはなりません。ところが模型関係の本では対象にしているのはラジコン模型の買い方や作り方やメインテナンス方法が主体で，プロポの詳細について触れているようなものは見当たりません。

しかたがありませんので，実際にプロポを入手して出力波形を調べてみることにします。入手したのは双葉電子工業の2チャンネルのラジコンカー用プロポである「FP-T2NBL」で，本体と箱に27 MHzのAMと書いてあるものです。入手は比較的容易であると思います。買いに行くときは27 MHzのAMということを必ず確認するようにしてください。昔はラジコンといえば27 MHz帯ばかりだったのですが，この周波数帯域はCBや漁業無線などと同じ周波数帯域であり妨害や混信が多いため，最近は40 MHz帯もよく使われるようになっています。定価はよくわかりませんが，何軒か店を回って調べた範囲では実売価格は1万円以下のようです。ちなみに，私が買った店では6,800円でした。

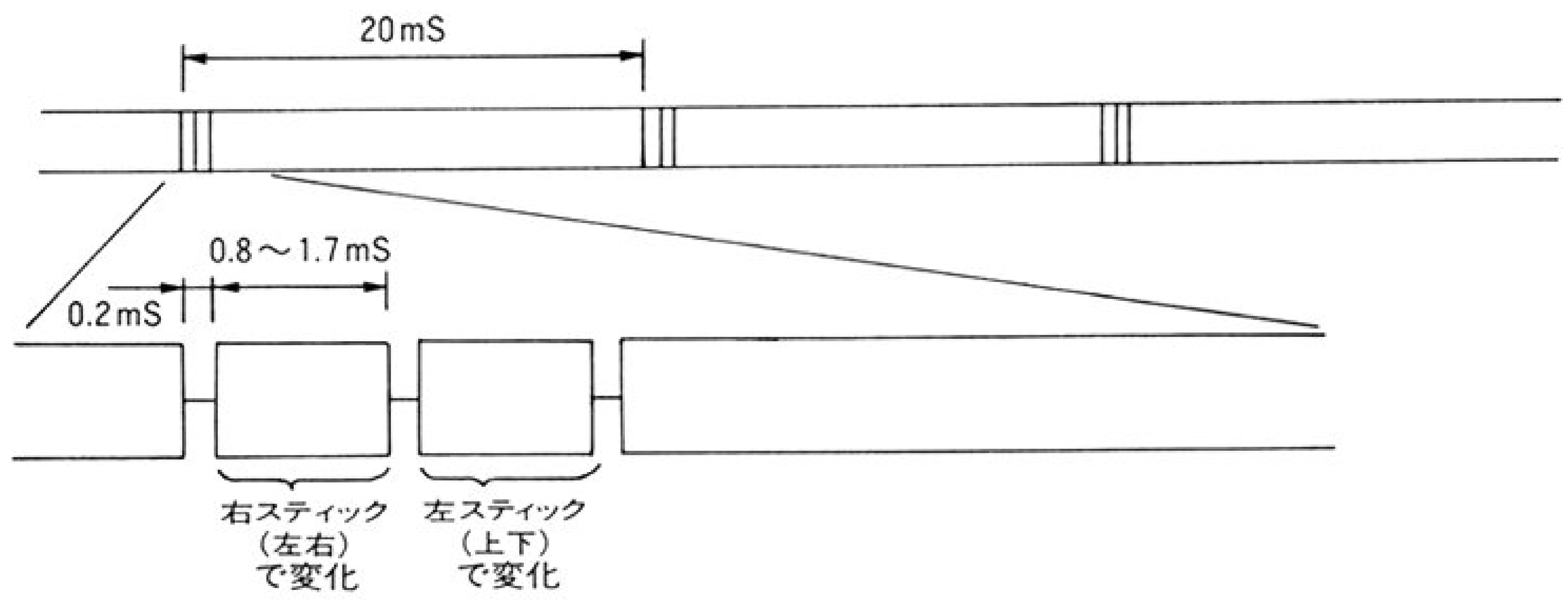
受信機やサーボモータは使用しませんので，もし手元に同じタイプのプロポをお持ちでしたらそのまま利用することができます。

●2　プロポはどうやって動く?

買ってきたFP-T 2 NBLは，箱を見る限り，27 MHzでAM方式であるということです。AMというのは信号にあわせて電波の強さを変えてやる方法のことで，振幅変調ともいわれます。普通の中波や短波のラジオは，この方法で音声を送っています。ちなみに，FMというのは周波数変調のことで，こちらは電波の強さは一定のまま，音の大小にあわせて周波数を変えるというものです。FMラジオやTVの音声などはFM方式です。

AMという以上，電波の強さの変化でデータを送っているのでしょう。試しにシンクロのプローブに小さなループアンテナをつけて，プロポのアンテナに近づけたら簡単に波形を観察することができました。振幅変調というよりは，27 MHzのON/OFF信号といったほうがよい

●図……1　プロポ（FP-T 2 NBL）の送信波形



ような波形です。さらにスティックを動かしながら波形を見ていたら，データの送り方は比較的単純であることがわかりました（図1参照）。

プロポからの電波は基本的に一定の強さで送信しっぱなしの状態になっていて，この中に一定周期で3回，電波が途切れるところが作られています。この途切れの間隔がスティックの傾きに対応しているのです。右のスティック（左右方向）を動かすと，最初の途切れたところから2番目のところまでの幅が変化します。2番目と3番目の間は左のスティック（上下方向）の傾きに応じて変化します。

ラジコンの受信側では，このパルスの幅を取り出し，それぞれのチャンネルのサーボモーターは自分の回転角と，与えられたパルスの幅との比較を行って誤差を修正する方向に回転し，一致すると停止するようになっているのでしょう。これを使ってスティックの傾きに応じて舵を切ったり，速度の調整をすることができるわけです。

整理すると，このプロポを使ったラジコンでは，次のような流れで情報が伝えられていることになります。

スティックの傾きの変化
｜
途切れと途切れの間隔の変化
｜（電波）
〈受信機〉
サーボモーターに与えるパルス幅の変化
｜
サーボモーターの回転

ラジコンスティックでは，この最終段であるサーボモーターへのパルスの幅を数値に直し，それをサイバースティックの仕様にあわせてX68000に渡せるようにすればよいということ

になります。

❷·1 波形の計測

次にプロポの送信波形をもう少し詳しく見てみます。まず，データを送っている周期ですが，これは約 20 mS とわかりました。1 秒に 50 回ほど送っているわけです。電波が途切れる時間は 0.2 mS です。スティックから手を離したまま，すなわちスティックが中央にある状態では，2 つある途切れと途切れの間隔（パルス幅）はどちらもだいたい 1.3 mS といったところです。

パルス幅は，左（あるいは上）に倒していくと狭くなり，最小で約 0.8 mS くらいまで狭くなります。逆に右（あるいは下）に倒すと，約 1.7 mS まで広がりました。

幅の変化の様子はわかったので，次にこの幅の変化と X68000 に与えるデータの関係をつかんでおきましょう。サイバースティックでは左（上）側が 00 H，右（下）側が FFH の方向でした。一方，プロポからのパルス幅は左（上）で狭く，右（下）で広いのです。単純にパルスの幅をカウンタで測定して，狭い側で 00H，広い側で FFH になるようにするだけでよさそうです。サイバースティックを設計した人がラジコンの事情を知っていたのか，単に VRAM の XY 座標にあわせただけなのかはわかりませんが，うまくしたものです。

スティックが中間の位置にあるときに測定された 1.3 ms というのは，0.8 ms と 1.7 ms の平均値程度であることから，特に小細工はしなくても問題なく，スティックが中央なら X68000 も中央にあるものとして受け取ってくれそうです。

●3 ラジコンスティックの回路設計

さて，プロポの動きがわかったところで，いよいよ回路の設計にとりかかることにします。少々複雑な回路になりそうですので，各回路ブロックごとに順に設計していくことにします。

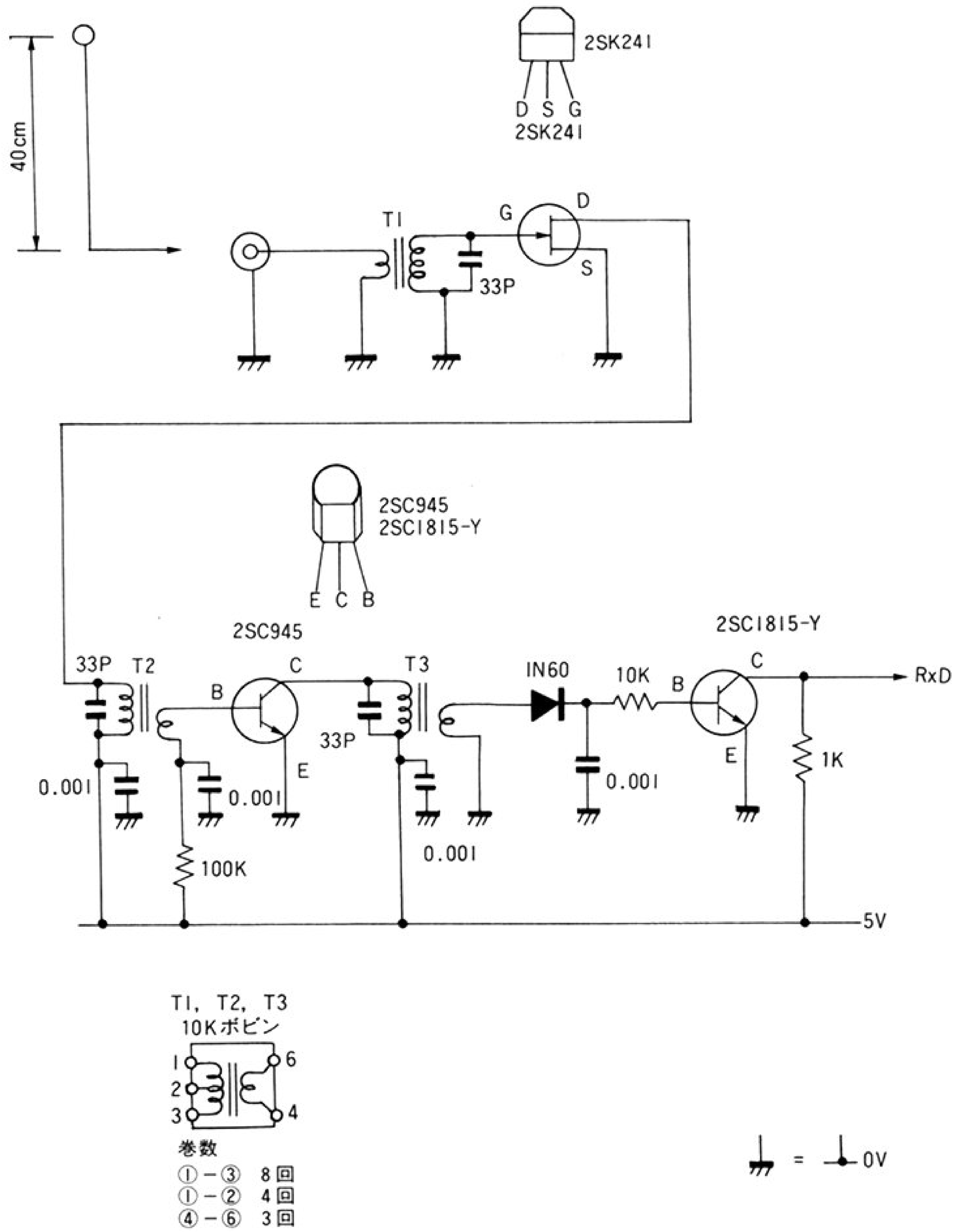
❸·1 プロポ受信回路の設計

まず，プロポからやってきた電波を受信し，デジタル IC が扱えるような信号に変換する必要

があります。買ってきたプロポには受信機がついていますが，自作品の中にブラックボックスがあるというのはあまりおもしろくないので自分で作ることにしました。

図２が受信部の回路です。この回路の出力は単純に受信した波形に応じたデジタル波形が得られます。プロポから電波が出ているときは０，途切れると１が出力されます。受信した電波はまずＴ１の同調回路で選択されます。コイルデータは回路図中に示しておきましたが，もし手に入るようでしたら FCZ 研究所のハムバンド用コイル(28 MHz 用)を使ってもかまいません。

●図……2　プロポ受信部の回路図



選択された電波は 2 SK 241 と次段の 2 SC 945 で増幅されます。増幅された電波は 1 N 60 で検波され，この出力を使って次段の 2 SC 1815 をスイッチングします。プロポからの電波がない状態ではトランジスタは OFF にしているので出力は 1 （High レベル） になっています。電波がくるとベース電流が流れるのでトランジスタが ON となり，出力が 0 になるわけです。

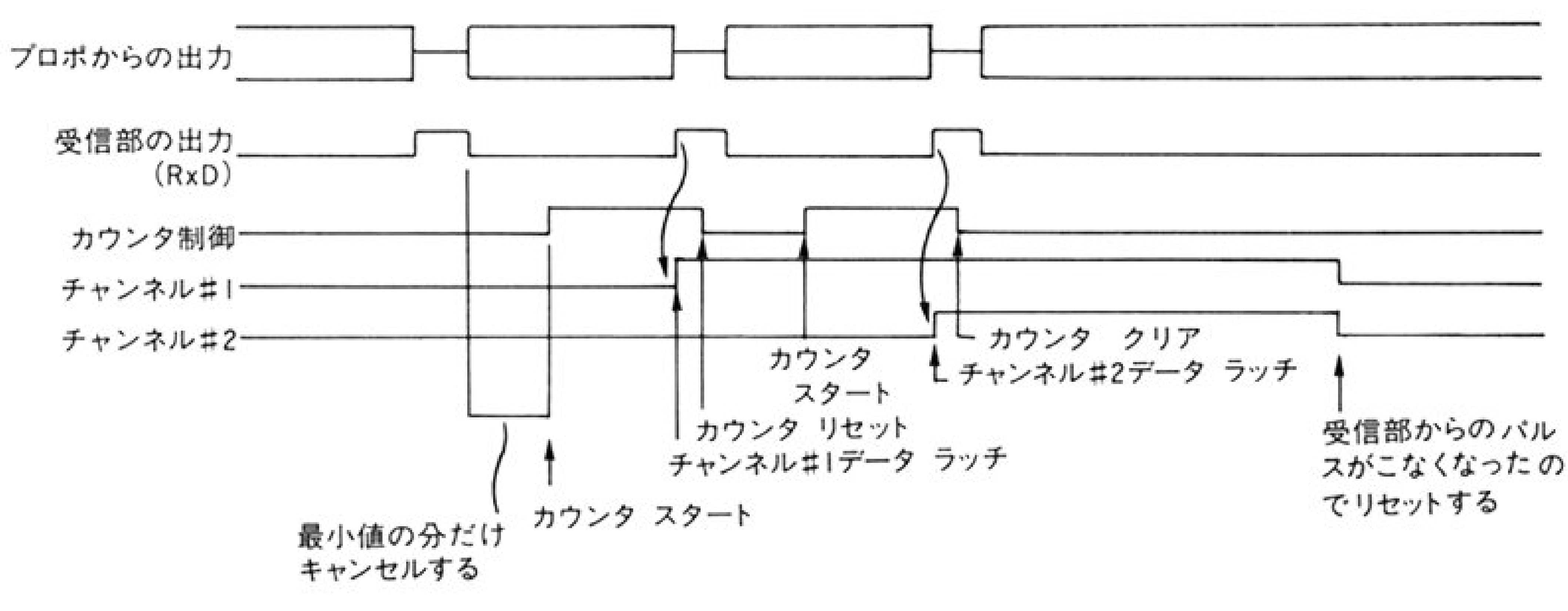
AGC をかけて検波出力を安定させたほうがよいかもしれません。ラジコン模型のように X 68000 が走り回るわけではありませんから出力レベルがそれほど大きく変化するとは考えられません。ここでは簡単に増幅，検波するだけですませることにしました。

❸·2 受信波形処理回路の設計

受信部からきたプロポの出力波形を受けて，パルス幅測定用のパルス信号を作成するのがこの部分の仕事です。受信回路から出力される波形はプロポからの電波がきていると 0，途切れると 1 になるようになっていますので，このパルス幅を数値化することになります。パルス幅，すなわち電波が途切れ，再び出始めたところから，次に途切れるまでの時間を計ればよいわけですが，単純にカウントしたのでは幅が最小のときに 0 にならなくなりますので，あらかじめ幅が最小のときの分を差し引いて測定します。図 3 は，この部分の考え方を示したものです。

途切れを検出したら，最小値だけ遅らせてカウンタをスタートさせます。次の途切れがきたら，その時点のカウンタの値をラッチします。少し遅らせてカウンタをリセットし，次のチャンネルのカウンタに備えます。

●図……3　受信波形からデータへの変換の考え方



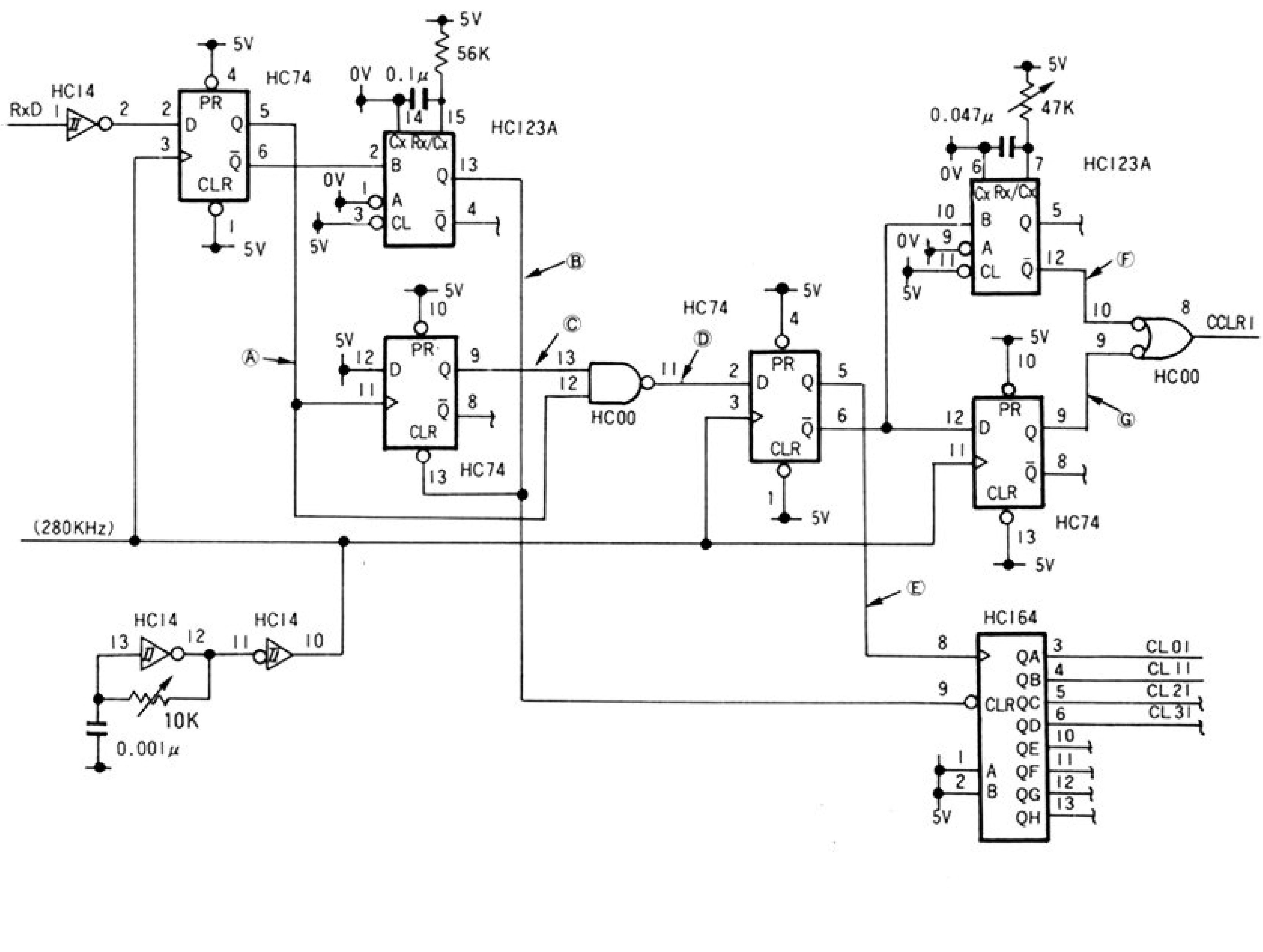
このままですと，どこかで途切れのカウンタを間違うと，上下と左右が逆になってしまい，以後はずっとずれたままになりますので，カウントをミスしても，つじつまをあわせる工夫をしておかなくてはなりません。今回は途切れが一定時間以上こなくなったことをつじつまあわせの鍵として使うことにしてみました。

図1を見てみるとわかるとおり，プロポの発信している波形は1回のデータ転送を行った後，次のデータを送ってくるまでには長い空き時間があります。これを使って，一定時間以上パルスがこなければ回路をリセットするようにしておけば，どこかでつじつまがあわなくなったとしても，次のデータからは正しく受け取れるようになるわけです。

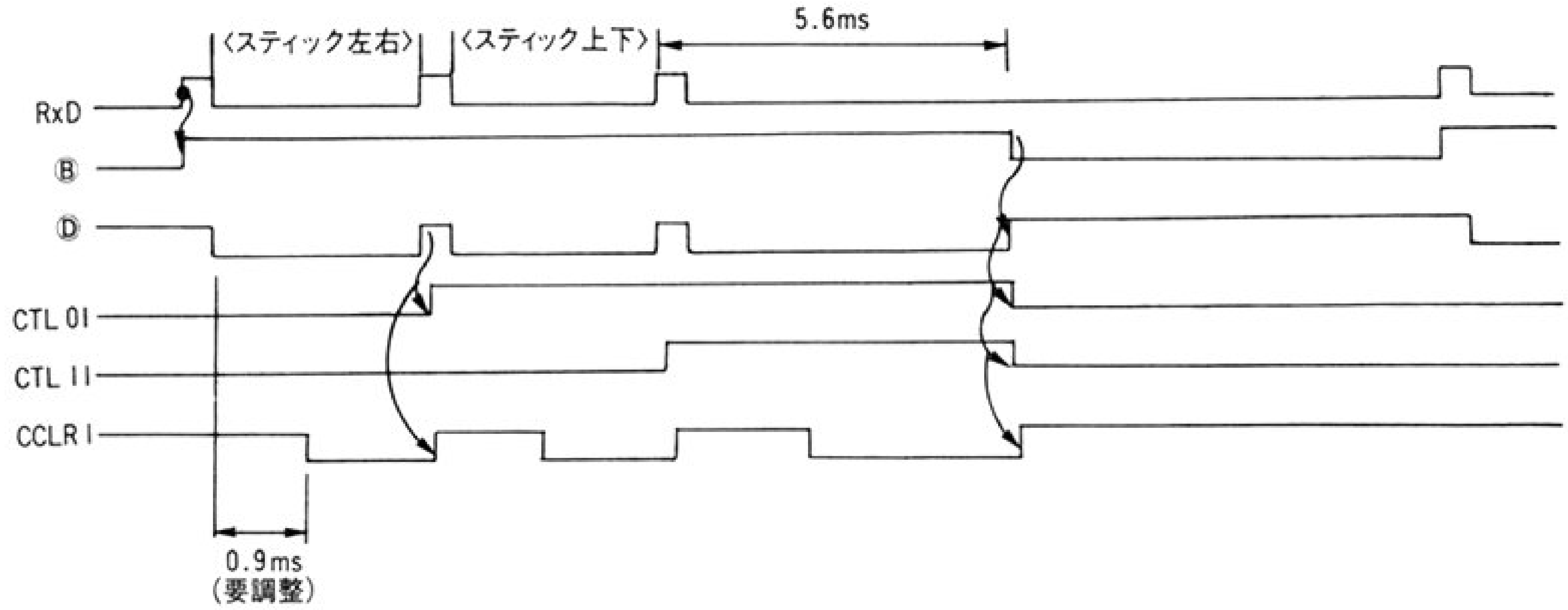
最終的な回路は図4のようになりました。図5，6に動作タイミング図を示しましたので，参考にしてください。この回路では受信回路からきた信号 RxD をもとにして，カウンタのリセット信号（CCLR1）とカウンタ値のラッチ信号（CL01～CL31）を発生しています。

今回使用したプロポは2チャンネルタイプですから，使用するのは CL01 と CL11 の2つだけです。CH21 と CH31 は，将来同じシリーズの4チャンネルプロポ（T4NBL など）が手に入ったときを想定したものです。

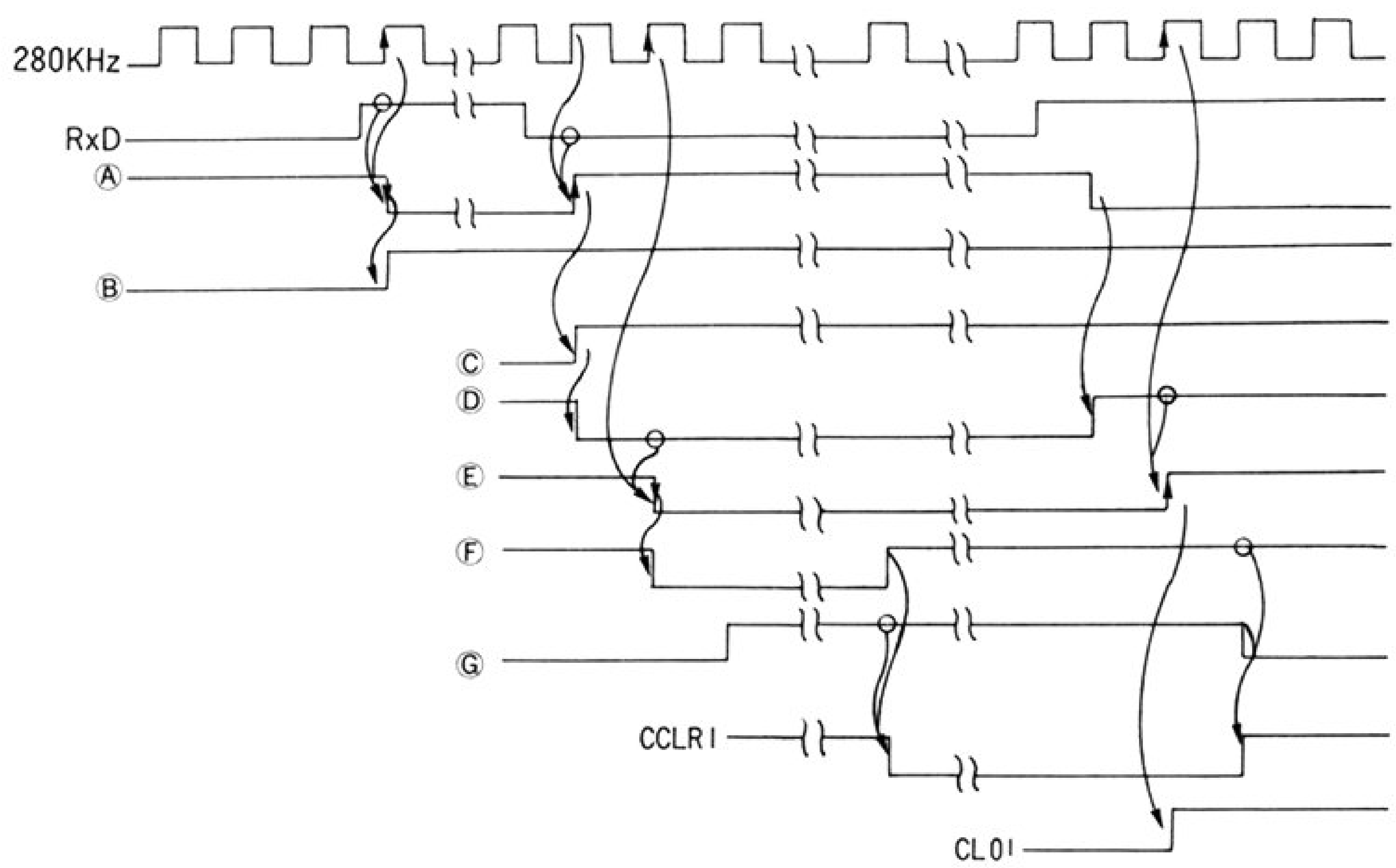
●図……4　受信波形処理回路



●図……5　受信波形処理回路の動作(1)



●図……6　受信波形処理回路の動作(2)



まず，左下のHC14が2つ並んでいるのが発振回路で，ここで作られた一定周期のクロックがラジコンスティック全体の動作タイミングを決めています。この周波数はプロポのパルス幅を使って調整します。プロポの送ってくるパルスの幅は最小時で約0.8mS，最大時で約1.7mSでした。

幅の変化は1.7−0.8=0.9mSです。これを256等分するクロックは0.9/256=0.00352mSの周期，つまり約280KHzとなります。HC 14を使った図のような発振回路では発振周波

数はだいたい 1/CR になります。ここでコンデンサを 0.001 μF とすれば，抵抗値は R=1/(280 K×0.001 μ) ですから，約 3.6 KΩ になります。パルスの幅はプロポの具合などによって変化しますので，それにあわせてクロック周波数を調整できるように 10 KΩ の半固定抵抗を使いました。

受信回路からきた波形は，いったん HC14 で波形整形した後，HC74 によって内部のクロックに同期させ，扱いやすくしておきます。その右側の HC 123 は，プロポからの途切れが一定時間続いているかを見るもので，約 5 mS の間，途切れがこなければ回路をリセットします。

パルスがきている間はカウンタを動かし，値をラッチさせる信号を作ります。HC123 の下の HC74 と，その隣の HC00 でちょっとした信号を作り，さらに HC74 で受けて 1 クロック遅らせます。

この右下にある HC123 は受信されたパルス幅から最小値のときの分をキャンセルするものです。この幅も調整できるようにしてあります。HC00 の出力が 0 になってからこの CR で決められる時間，HC123 の出力は 0 になっているため，CCLR 1 が 1 になり，カウンタはリセットされたままになります。HC 123 の出力が 1 に戻ると，カウンタが計数を開始するわけです。HC00 の出力の立ち上がりがカウントの終了を示すことになります。順序として，まずカウンタにラッチ信号を送るため，HC164 にこの波形を与えます。さらに，次のチャンネルのカウントに備えるため，カウンタのリセット信号をラッチ信号から 1 クロック遅らせて送ります（HC 164 の上にある HC74)。

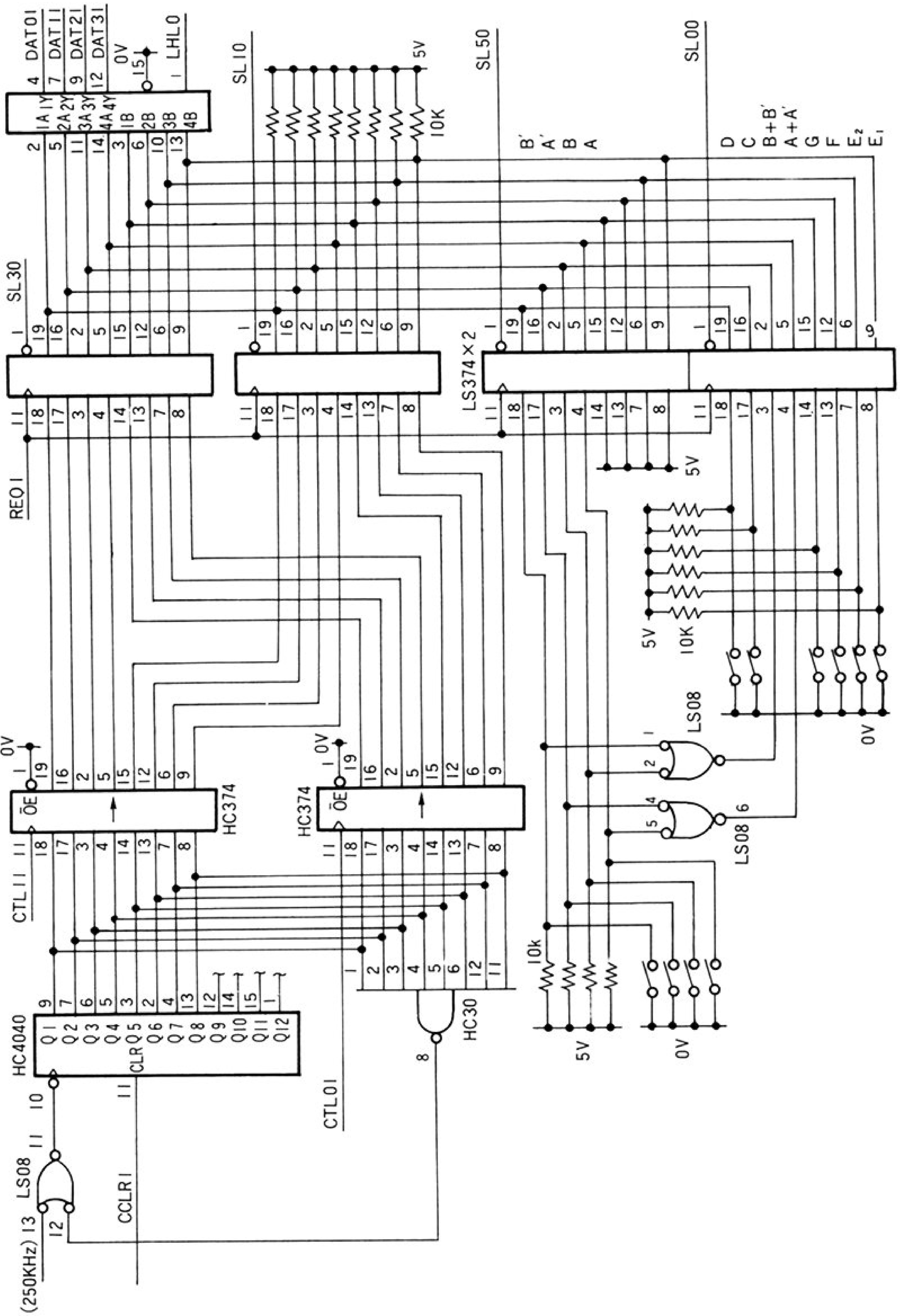
HC00 と HC164 は直結してもよさそうに見えます (たぶん，直結しても大丈夫でしょう)。ここでわざわざ HC74 で遅らせたのは，HC74 が 1 ゲート余っていたということと，HC123 による時間監視でリセットがかかったときに HC00 の出力は即座に 1 になるために HC164 にはクロックとリセットが同時(といっても，HC00 の分だけ遅れますが)に入ることになるのが気持ち悪かったからです。

❸·3 カウンタ回路の設計(1)

次に，受信波形回路で作った波形に従ってパルス幅をカウントし，データをラッチする回路を見ておきます。図 7 の左半分がこれにあたります。

HC 4040 は 12 ステージという，段数の多いカウンタです。普通の 74 LS シリーズでは，これほど大きい段数のものはありません。今回は，このカウンタの下位 8 ビットを使用します。HC4040 の下にある HC30 は，カウンタの値が FFH になったら，それ以上進まないようにするためのものです。カウンタはクロックの立ち下がりで進むタイプなので，ここが FEH から FFH になったとき，クロックは 0 (Low レベル) になっています。そこで，FFH になったら

●図……7　カウンタ周りの回路



HC4040のクロック入力を強制的に0に固定してしまうわけです。これでカウンタにはクロックが入らなくなり，カウントはFFHで停止します。カウンタにクリア信号が入るとカウント値が00Hになり，HC30の出力も1になり，再びクロックが入ってくるようになります。

❸·4 ホストインタフェース回路の設計

カウンタ回路の右半分にいくまえにホスト，つまりX68000とのインタフェース部分の回路設計を行ってしまいましょう。図8にこの部分の回路を示します。

このブロックはホストからのREQ信号を受けて動作を開始し，サイバースティックに似せたタイミングでホストにカウンタやスイッチの値を渡す部分です。サイバースティックではこの処理をワンチップマイコンを使って行っているようですが，自作品でマイコンを使うとROMライタを作ったり，プログラムを作るなどの手間がかかることから，今回はランダムロジックで作ることにしました。

サイバースティックでは転送速度を何通りか選べるのですが，遅いモードというのはCPUの能力が低かったり，不定期にDMA転送が入り込んだりするために最速モードではついてこられないコンピュータ向けに作られたものです。X68000なら最速モードだけで十分ですから，今回は最速モードでの動作のみとします。もし遅いホストマシンにつなぐ場合は，このブロックに入っているクロック（280 KHz）だけを半分なり，1/3なりにすればそれだけ転送タイミングが遅くなります。

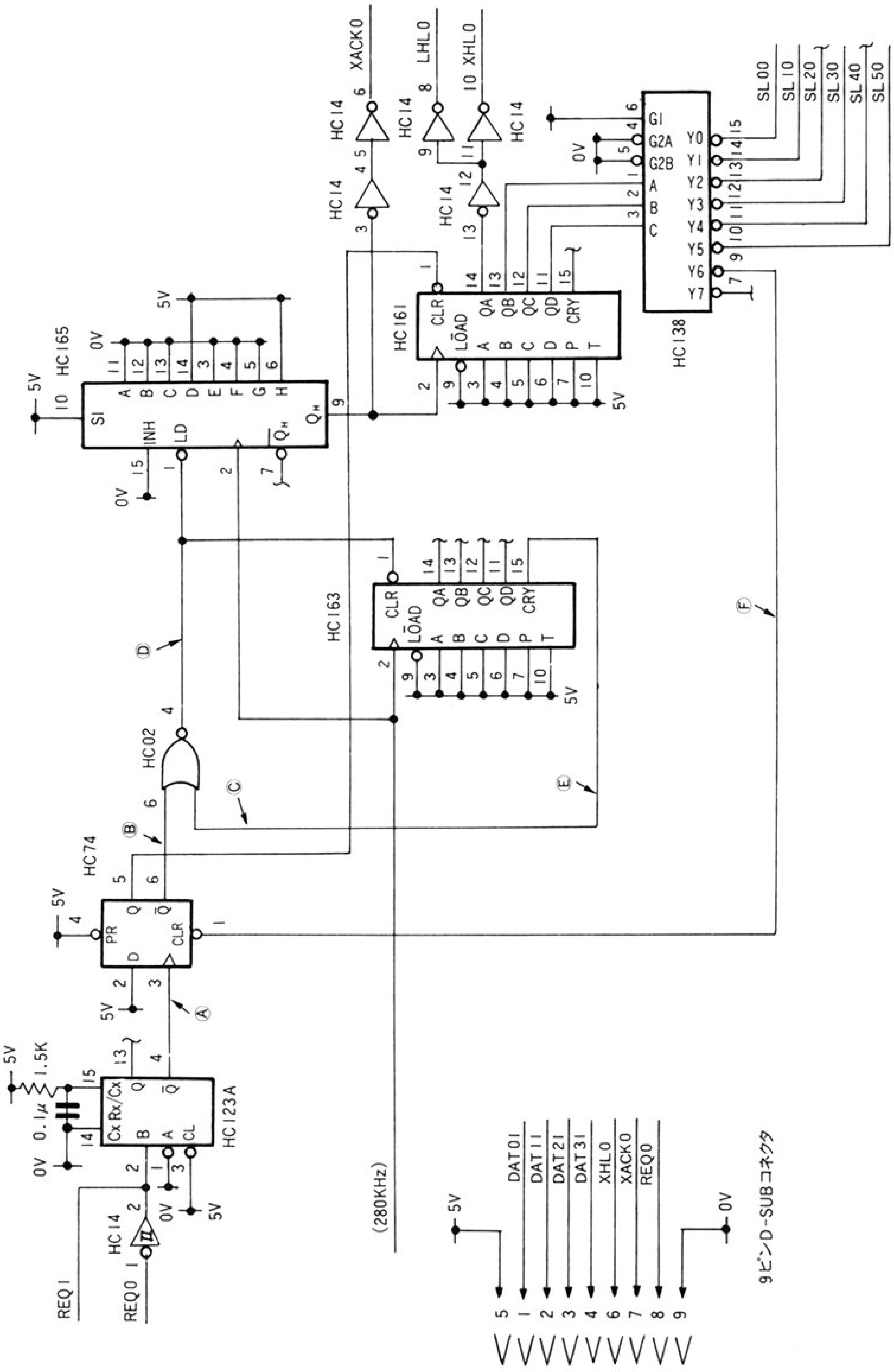
図9，10が動作タイミング図です。図9はデータ転送全体を見たところ，図10は1回（1組）の転送のタイミングを拡大したものです。図11は転送しているデータのフォーマットです。

ホストと接続される信号はREQ0, XACK0, XHL0, そしてカウンタ回路のDAT01～DAT31の計7本です。

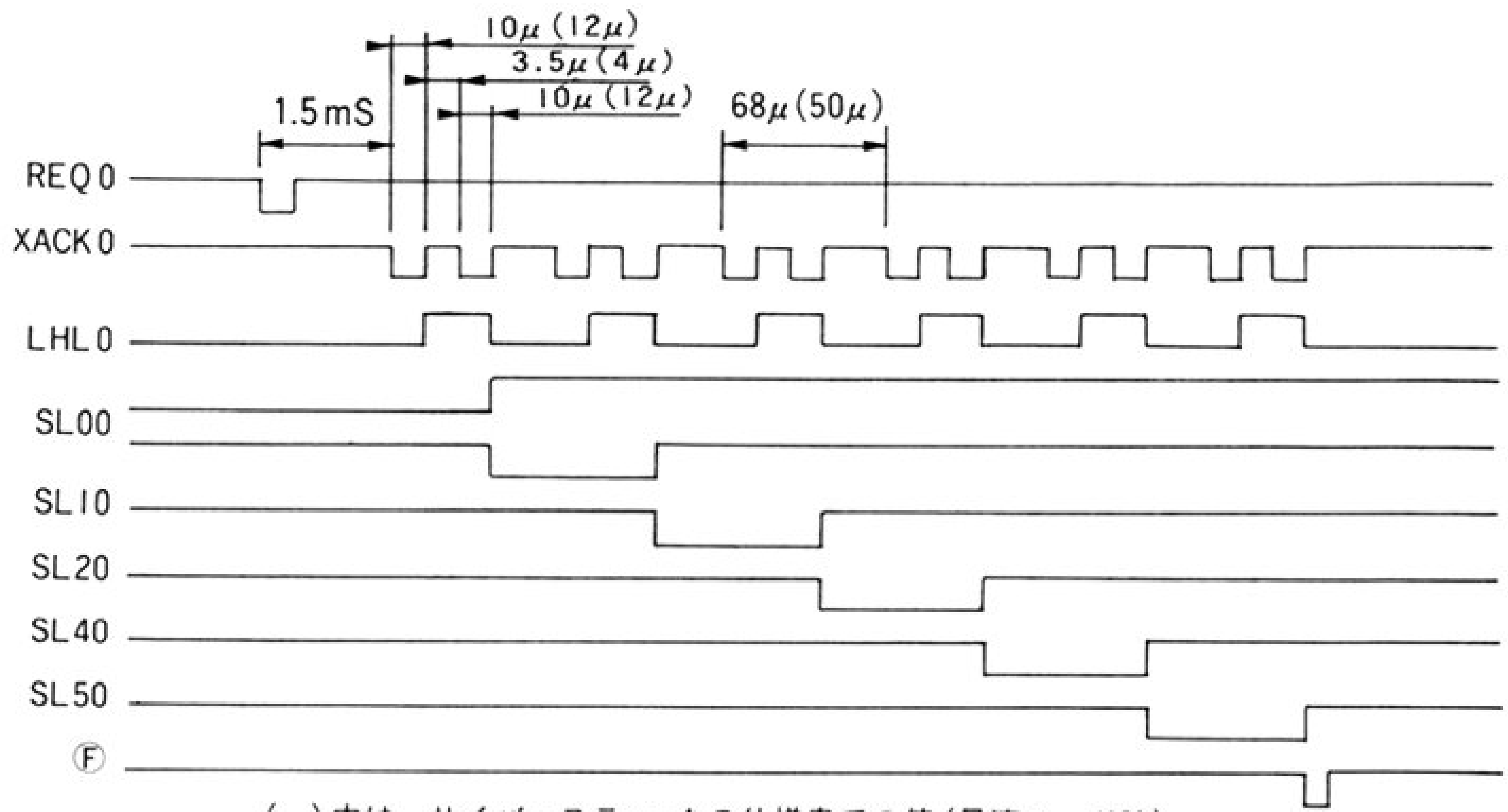
動作タイミングの作成のための基準クロックには，波形処理回路で作った約280 KHzのクロックをそのまま使っています。このクロックはスティックからの受信データによって調整することになっていますから，それに応じてホストインタフェース回路のタイミングも変化することになります。これを気持ち悪いと思われる方もおられるでしょうが，現実にはホスト側のタイミング管理はかなりいいかげんです。サイバースティックのドライバのソースリストを見てもわかるように，要は規定時間以内（これもまたけっこう長い時間なのですが）に送り終わればそれでよいということになっていますから基本クロックが多少変化しても大した問題にはなりません。図9にクロックを280 KHzとしたときのタイミングとサイバースティックの仕様書上の値を書いておきましたので参考にしてください。

それではホストインタフェース回路の動作を見ていきましょう。回路自体はそれほど複雑な

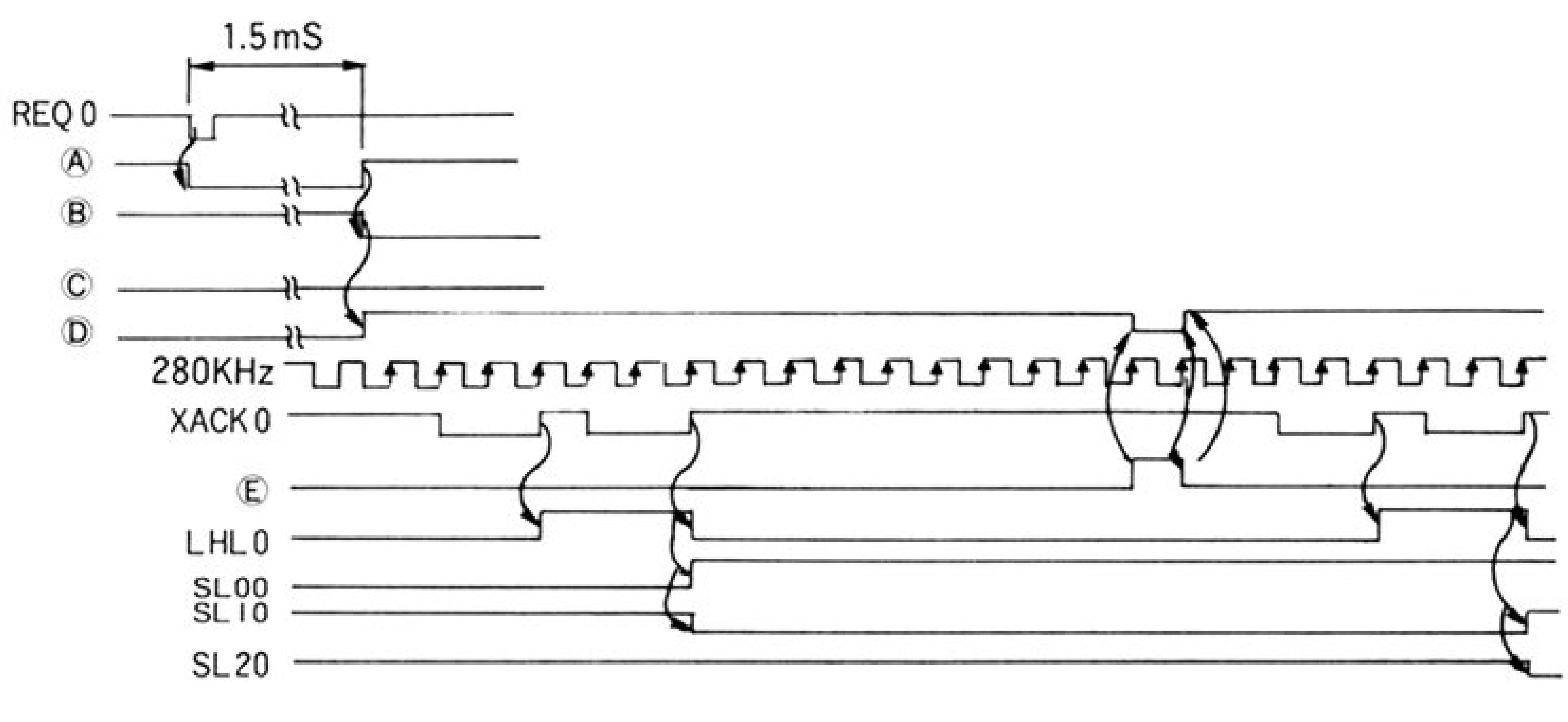
●図……8　ホストインタフェースの回路図



●図……9　ホストインタフェースのタイミング図(1)



●図……10　ホストインタフェースのタイミング図(2)



ものではありません。左端の REQ 0 というのが X68000 からの転送開始要求で，これが 0 になった時点からタイミングがスタートします。いきなり HC 123 のワンショットでタイミングを遅らせています。

当初，このワンショットはつけていなかったのですが，サイバースティックのタイミングを調べているとき，REQ を 0 にしてから 1.5 mS 程度遅れてデータの転送が始まっていることがわかったので，まねをするために入れています。

●図……11　サイバースティックのデータフォーマット

回	XHL	DAT31	DAT21	DAT11	DAT01
1	0	A+A′	B+B′	C	D
2	1	E_1	E_2	F	G
3	0	チャンネル#1 上位			
4	1	チャンネル#2 上位			
5	0	チャンネル#3 上位			
6	1	チャンネル#4 上位			
7	0	チャンネル#1 下位			
8	1	チャンネル#2 下位			
9	0	チャンネル#3 下位			
10	1	チャンネル#4 下位			
11	0	A	B	A′	B′
12	1	〔幻の4ビット〕			

- スイッチは押すと0，離すと1になる
- A＋A′(B＋B′)はA(B)とA′(B′)のいずれかが押されていれば0になる
- サイバースティックではFはSTART，GはSELECTボタンになっている
- アナログデータは各チャンネルとも8ビットあり，4ビットずつに分割して送られる

次のHC74は，データ転送中を示す信号を作るものです。REQがくると，HC123の分だけ遅れてこれがセットされ，全データの転送が終わると，リセットされます。図面中央のHC163は2回のデータ転送ごとに少し時間をとるものです。サイバースティックでは，データの転送は2回を1組として送るようなタイミングになっています。この2回分のデータ転送のACK(X68000への応答信号)を作っているのが右上のHC165です。

ACKの回数は，その下のHC161でカウントされ，最下位が偶数番目か，奇数番目かを示すビットに，そして2ビット目以降で何組目の転送なのかを示すのに使われます。これがHC138で分解されるわけです。12回の転送が終わるとY6が0になり，左上のHC74がリセットされ，回路全体が初期化され，転送終了となります。

❸・5 カウンタ回路の設計(2)

さて，先ほどやり残しておいたカウンタ回路(図7)の右半分を見ておきましょう。左半分のラッチ（HC374）にはプロポが送ってきたスティックの傾きのデータが入っています。このデータをホストからのREQ信号がきたときに右のラッチに取り込みます。こうしておかないと，ホストにデータを送っているうちにデータが変化する恐れがあるからです。

配線がクロスしているのは，ホストへの転送順序がまず各チャンネルの下位ニブル（4ビット単位のデータ）をすべて送った後で上位ニブルを送るようになっているのに対して，この回路では1つのラッチICの下位ニブル，上位ニブルという順序で送っているためです。

スイッチのほうもデータをラッチさせています。SL50という信号がつながっているLS374の入力4本が5Vにつないであります。これがサイバースティックのマニュアルでは記述されていない幻の12回目の転送データになっています。ここには他の入力と同じようにスイッチなどをつけてホストのソフトで対応してやればちゃんと使うことができます。

この回路ブロックでは，ラッチアップ対策を手抜きするためにスイッチ周りだけLSシリーズを使いました。作るときに間違えないようにしてください。

●4 ラジコンスティックの製作

特に入手しにくい部品はないと思いますが，あえていうならスイッチがちょっと悩むところかもしれません。押したときにONになるようなスイッチであれば何でもかまいません。私は秋葉原のヒロセパーツセンターの入口のところに並べられていたジョイスティック用のスイッチを買ってきて並べたのですが，やってみると両手はプロポにとられているので手では押すことができません。しかたなく外部スイッチ端子をつけてフットスイッチをつけられるようにしてみました。フットスイッチはオートメーションパーツ屋で入手したものを使っていますが，楽器屋で電子楽器に使うものを入手しても同じように使えると思います。

ケースや電源ランプ，アンテナ端子などは各自の趣味で選んでください。スイッチもゲーム用に限定するのであれば2つあれば十分でしょう。

製作は受信回路から片付けてしまいましょう。この部分はかりにも高周波を扱いますから，グランド（0V）のパターンをなるべく広くとるようにします。サンハヤトの504EGを使い，

カッターでパターンを切り取って必要なパターンを作り，残った部分をすべてグランドとして使うようにしてみました。部品配置などは，信号の流れどおりにすればよいでしょう。

デジタル部のICの配置は信号の流れを考えながら決めていきます。ICの5VとGND（0V）の間にはIC1個，ないし2個に1つずつくらい0.1μFくらいのセラミックコンデンサをつけておいてください。74HCシリーズに限らず，C-MOSのICは動作時と非動作時の電流の変化が74LSシリーズなどのTTLに比べて大きく，電源にノイズが乗りやすいので，コンデンサはまめにつけてください。

各ICの空きピンのうち，入力になっているものは5Vか0Vのいずれかに接続して，入力がふらつかないようにしておいてください。また，私の作ったものは各回路ブロックごとに基板を分けていますが，これは回路を製作/調整するときに各ブロックごとにやっていたときの名残です。これから作られる方は1枚の基板にまとめたほうが基板間のつなぎの線がはい回ることもなくてよいでしょう。

●5 調　整

でき上がったら，まず電源ラインをテスターでチェックしておきます。ショートはないか，電源の配線を間違えていないか，念入りにチェックしておきましょう。実験用の電源を持っていればそれで5Vを，なければ十字を切ってX68000のジョイスティック端子につないで（過大電流が流れるとX68000の電源が自動的に出力をカットしますから，本体が壊れる心配はありませんが）電源スイッチON！　まともなら，まず受信回路のコイルの調整からかかります。金属のドライバは使えません。プラスチックでできたコアドライバと呼ばれるものを持っていればいちばんよいのですが，なければマッチ棒やいらなくなった耳かきの先を削ったものなどでもなんとかなります。

アンテナ端子に10cmくらいの電線をつけておきます。IN60のカソード側（コンデンサのある側）の電圧をテスターで測りながらプロポの電源を入れます。プロポの電源を入れると電圧が少し増えるはずです。この状態でコイルの中にあるコアを回していって，振れが最大になるところを探します。コアが入ったところと出たところの2カ所でピークがあると思いますが，コアが入った側で固定します。3つのコアを最良の位置にすると，最初のときからは信じられないくらい大きな電圧になっているはずです。

ここで，プロポのスイッチを切ります。もし針が戻らないようなら，受信回路が発振してい

ます。コアを最高点からずらして発振しないようにしてください。受信している電波と同じ周波数で2回も増幅しているので，やや発振しやすいのは事実です。

次にデジタル部の調整にかかります。リスト1のアセンブラで書いた外部関数とリスト2のBASICプログラムを入力してください。基板上の半固定抵抗はとりあえず真ん中あたりにしておきます。プログラムを走らせると，読み取ったデータが表示されます。「入力エラー」の表示が出るようなら，ホストインタフェース回路周りをチェックしてください。なんらかの入力データが表示されたら，まずはスイッチ入力が行われるかチェックします。押したスイッチが正しく読めればホストインタフェースはとりあえず正しく動いていると考えられます。

ホストインタフェースがうまく動いたら，次にカウンタ周りの調整にとりかかることにします。プロポのスイッチを動かしてデータが変化することを確認します。スティックのトリムレバーは真ん中にセットしておきます。データが多少グラグラするのはしかたないようです。

スティックを最小値側に倒します。最小値カット幅調整用の半固定抵抗を回して，カウントが進む少し前にあわせておきます。次にスティックから手を離し，センター位置にします。この状態で読み取り値が80H近辺になるように発振周波数調整の半固定抵抗を調整します。スティックを値が最大になる側に倒して読み取り値がFFHになるかチェックします。ならないようなら，発振数を引き上げ，今度はセンター位置で80Hになるように最小値カット幅調整用半固定抵抗を調整します。これを何度か繰り返していい感触のところに追い込んでいきます。チャンネルごとの値のばらつきはプロポ側のトリムレバーで調整できますので，それほど神経質にならなくても大丈夫です。

スティックのトリムレバーは真ん中にセットしておきます。

シンクロスコープがあれば調整はかなり楽になるでしょう。スティック位置が最小値側のときに波形を見ながら最小値カット用半固定抵抗を調整し，そのうえで発振周波数を調整して，センターで80H付近になるようにすればほぼ完了です。

6 使用感

調整の終わったラジコンスティックが実際にどの程度実用になるものか，アフターバーナーで試してみました。スロットルに相当するレバーがないので常時フルスロットルになってしまうのはちょっと問題ですが，操作感自体はかなり良好です。やはりアナログスティックの歴史では一日の長のあるラジコンのプロポです。スティックの長さやバネの堅さ，スティックの位

置など，なかなかうまくできています。

●7 おわりに

サイバースティックのインタフェースを利用してラジコンのプロポと接続するという当初の目的は十分達成されました。インタフェース上でも幻のアナログチャンネル，お化けの 12 回目の 4 ビットデータなど，サイバースティックの隠された機能もみつかりました。今回の回路のうち，ホストインタフェースを流用すれば，A/D コンバータでも何でもつなぎたい放題です。幻のチャンネルも有効に使えるでしょう。

アナログチャンネルもすべてスイッチとして使ってしまえば，なんと 48 個のスイッチが読めることになります。通常のジョイスティックなら 8 個分です。ジョイスティックポートを 2 つとも使って 16 人同時プレイの対戦ゲームなどにするのもおもしろいかもしれません。

●リスト……1　ラジコン・スティックチェックプログラム

```
1000 /*-----------------------------------------*/
1010 /*- ラジコン・スティックチェックプログラム  */
1020 /*-                                       -*/
1030 /*- 1990-02-11 written by M.kuwano        -*/
1040 /*-                                       -*/
1050 /*-    No rights reserved.                -*/
1060 /*-                                       -*/
1070 /*-----------------------------------------*/
1080 char c(5)
1090 int stat,px,py,pz,pt
1100 px=0:py=0
1110 pr_info()
1120 astset()
1130 repeat
1140     stat = astick(c)
1150     disp_val(stat)
```

```
1160    disp_pos()
1170 until inkey$(0)=" "
1180 astrst()
1190 end
1200 func disp_val(stat;int)
1210    int i
1220    locate 0,9
1230    if stat<>0 then color 2: print"ERROR" else color 3:print"READY"
1240    locate 0,8
1250    for i=0 to 3
1260        print right$("0"+hex$(c(i)),2);"  ";
1270    next
1280    i=&H80
1290    repeat
1300        pr_onoff(c(4) and i)
1310        i=i/2
1320    until i=0
1330    i=&H80
1340    repeat
1350        pr_onoff(c(5) and i)
1360        i=i/2
1370    until i=0
1380 endfunc
1390 func pr_onoff(s;int)
1400    if s=0 then print"ON   "; else print"OFF  ";
1410 endfunc
1420 func pr_info()
1430    screen 2,0,1,1
1440    console ,,0
1450    color 3
1460    locate 0,5:print"終了する時はスペース・バーを押してください"
1470    color 1
1480    locate 0,7
1490    print"#1  #2  #3  #4  A+A' B+B'  C    D    E1   E2   F    G    ?    ?    ?    ?    A    B    A'   B'"
1500     box(349,239,349+257,239+257,15)
1510     box(49,239,49+257,239+257,15)
1520 endfunc
1530 func disp_pos()
1540     line(350,240+py,350+255,240+py,0)
```

```
1550      line(350,240+c(0),350+255,240+c(0),15)
1560      py=c(0)
1570      line(350+px,240,350+px,240+255,0)
1580      line(350+c(1),240,350+c(1),240+255,15)
1590      px=c(1)
1600      line(50,240+pz,50+255,240+pz,0)
1610      line(50,240+c(2),50+255,240+c(2),15)
1620      pz=c(2)
1630      line(50+pt,240,50+pt,240+255,0)
1640      line(50+c(3),240,50+c(3),240+255,15)
1650      pt=c(3)
1660 endfunc
```

●リスト……2　アナログ・ジョイスティック読み込み関数

```
*------------------------------------------------------------
*-      アナログ・ジョイスティック（サイバー・スティック）     -
*-      読み込み関数                                          -
*-                                                            -
*-      1989-06-17 M.Kuwano, No rights reserved               -
*-      1990-02-11 デバッグ＆１２回転送対応に変更             -
*............................................................
*-      呼び出し方（例）：                                    -
*-      char c(5)                                             -
*-      astick(c)                                             -
*-      入るデータは・・                                      -
*-              c(0).....右側スティック左右                   -
*-              c(1).....右側スティック前後                   -
*-              c(2).....左側スティック前後（スロットル）     -
*-              c(3).....予備スティック（将来用？）           -
*-              c(4).....トリガ・ボタン                       -
*-              c(5).....トリガ・ボタン                       -
*-      と、なります                                          -
*-      配列は６バイト以上確保してください。                  -
*-      配列のサイズチェックをやっていないので、あらぬところまで-
*-      書き込んでしまいます。                                -
*............................................................
*-      いつもながらインタプリタ／コンパイラ共用です          -
*-                                                            -
```

```
*--------------------------------------------------------------
                .include        doscall.mac
                .include        fdef.h
                .globl          _astick
                .globl          _astset
                .globl          _astrst

IOCS            equ             $0f

PPI_PORT_A      equ             $e9a001
PPI_PORT_B      equ             $e9a003
PPI_PORT_C      equ             $e9a005
PPI_CWR         equ             $e9a007

RQ_ASSERT       equ             $8
RQ_NEGATE       equ             $9

TIME_LIMIT1     equ             1000
TIME_LIMIT2     equ             100
                .text
                .even
*
*  インフォメーション・テーブル
*
*
                .dc.l           AS_INIT
                .dc.l           AS_RUN
                .dc.l           AS_END
                .dc.l           AS_SYS
                .dc.l           AS_BRK
                .dc.l           AS_CTRL_D
                .dc.l           AS_RES1
                .dc.l           AS_RES2
                .dc.l           PTR_TOKEN
                .dc.l           PTR_PARAM
                .dc.l           PTR_EXEC
                .dc.l           0,0,0,0,0

AS_RES1:
AS_RES2:
```

```
AS_END:
AS_BRK:
AS_CTRL_D:
AS_INIT:
AS_RUN:
AS_SYS:
                rts

*
*  トークン・テーブル
*
PTR_TOKEN:
                .dc.b           'astick',0
                .dc.b           'astset',0
                .dc.b           'astrst',0
                .dc.b           0


                .even
*
*  パラメータ・テーブル
*
PTR_PARAM:
                .dc.l           ASTICK_PAR
                .dc.l           ASTSET_PAR
                .dc.l           ASTRST_PAR

*
*  パラメータＩＤテーブル
*
ASTICK_PAR:
                .dc.w           aryl_c
                .dc.w           int_ret
ASTSET_PAR:
                .dc.w           void_ret
ASTRST_PAR:
                .dc.w           void_ret

*
*  関数アドレステーブル
*
```

```
PTR_EXEC:
                .dc.l           astick
                .dc.l           _astset
                .dc.l           _astrst

*
*  スタック・バッファ
*
SPBUF:
                .ds.l           1

                .even
*
*  アナログ・ジョイスティック読みだし（インタプリタ用）
*
astick:
                movea.l         12(sp),a1
                lea             10(a1),a1
                move.l          a1,-(sp)
                bsr             _astick
                addq.l          #4,sp
                moveq.l         #0,d0
                rts

*
*  アナログ・ジョイスティック用に、ＰＣ４を１にする
*
_astset:
                clr.l           -(sp)                   *       SPBUF = _SUPE
R(0);
                dc.w            _SUPER
                addq.l          #4,sp
                move.l          d0,SPBUF
                movea.l         #PPI_CWR,a0             *       *ppi_cwr = RQ
_NEGATE;
                move.b          #RQ_NEGATE,(a0)
                move.l          SPBUF,d0                *       _SUPER(SPBUF)
;
                bmi             astset_already_super
                move.l          d1,-(sp)
```

```
		dc.l		_SUPER
		addq.l		#4,sp
	astset_already_super:
		moveq.l		#0,d0
		lea.l		AS_RETVAL,a0
		move.w		d0,2(a0)
		rts

*
*  プログラム終了後、ＰＣ４を０に戻しておかないと
*  デジタル・モード用のソフトのうち動かなくなるも
*  のがでてくるらしい
*
_astrst:
		clr.l		-(sp)			*	SPBUF = _SUPE
R(0);
		dc.w		_SUPER
		addq.l		#4,sp
		move.l		d0,SPBUF
		movea.l		#PPI_CWR,a0		*	*ppi_cwr = RQ
_ASSERT;
		move.b		#RQ_ASSERT,(a0)
		move.l		SPBUF,d0		*	_SUPER(SPBUF)
;
		bmi		astrst_already_super
		move.l		d1,-(sp)
		dc.l		_SUPER
		addq.l		#4,sp
	astrst_already_super:
		moveq.l		#0,d0
		lea.l		AS_RETVAL,a0
		move.w		d0,2(a0)
		rts

*
*  アナログ・ジョイスティック読みだし（コンパイル時用）
*
_astick:
		bsr		get_astick		* get_astick();
		lea.l		AS_RETVAL,a0
```

```
		move.l		d0,6(a0)
		move.l		4(sp),d0
		move.l		d0,-(sp)
		bsr		aj_compile
		addq.l		#4,sp
		lea.l		AS_RETVAL,a0
		move.l		6(a0),d0
		rts

*
* アナログ・ジョイスティックデータ取り込み
*
* a0 Buffer_pointer
* a1 PPI_PORT_A
* a2 PPI_CWR
* d0 data
* d1 data
* d2 Loop counter
* d3 Timeout counter
*
get_astick:						* get_astick() {
		clr.l		-(sp)			*	SPBUF = _SUPE
R(0);
		dc.w		_SUPER
		addq.l		#4,sp
		move.l		d0,SPBUF
		move.w		sr,-(sp)		*	PUSH(SR);
		ori.w		#$0700,sr		*	disable_trap
();
		lea.l		AS_TMP_BUF,a0		*	buffer_pointe
r = AS_TMP_BUF;
		movea.l		#PPI_PORT_A,a1		*	ppi_port_a
  = PPI_PORT_A;
		movea.l		#PPI_CWR,a2		*	ppi_cwr
  = PPI_CWR;

		move.w		#5,d2			*	loop_counter
= 5;
		moveq.l		#0,d0			*	joydata = 0;
		move.w		#TIME_LIMIT1,d3		*	timer = TIME_
```

```
LIMIT1
        _astick_0:                                      *   do {
                move.b          #RQ_ASSERT,(a2)         *       *ppi_cwr
= RQ_ASSERT
        _astick_1:
                move.b          (a1),d0                 *       while ((
(data = *ppi_port_a & 0x60) != 0) && timer--)
                move.b          d0,d1                   *               ;
                andi.b          #$60,d0
                dbeq            d3,_astick_1
                bne             _astick_timeout         *       if (!time
r) goto _astick_timeout;
                move.b          d1,(a0)+                *       *buffer_p
ointer++ = data;
                move.b          #RQ_NEGATE,(a2)         *       *ppi_cwr
= RQ_NEGATE;
        _astick_11:
                btst.b          #5,(a1)                 *       while(!
(*ppi_port_a & 0x20) && timer--)
                dbne            d3,_astick_11           *               ;
                beq             _astick_timeout         *       if (!tim
er)     goto _astick_timeout;
        _astick_12:                                     *       while((
(data = *ppi_port_a) & 0x40) && timer--)
                move.b          (a1),d1
                btst            #6,d1                   *               ;
                dbeq            d3,_astick_12
                bne             _astick_timeout         *       if (!tim
er) goto _astick_timeout;

                move.b          d1,(a0)+                *       *buffer_
pointer++ = data;
                dbra            d2,_astick_0            *   } while (loop_
counter--);


                moveq.l         #0,d3                   *   retstat = 0;
        _astick_exit:
                move.w          (sp)+,sr                *   sr = POP(); /
* 割り込みフラグを戻す  */
```

```
                move.b          #RQ_NEGATE,(a2)         *     *ppi_cwr = RQ
_NEGATE;
                move.l          SPBUF,d1                *     _SUPER(SPBUF)
;
                bmi             _astick_already_super
                move.l          d1,-(sp)
                dc.l            _SUPER
                addq.l          #4,sp
        _astick_already_super:
                move.l          d3,d0
                rts                                     * }

        _astick_timeout:
                moveq.l         #1,d3                   *     data = 1;
                bra             _astick_exit
*
* アナログ・ジョイスティックデータ編集
*

aj_compile:                                     * aj_compile(pack_data) {
                movea.l         4(sp),a0
                lea.l           AS_TMP_BUF,a1

                move.b          2(a1),d1        *     pack_data[0] = joy_ra
w_data[6] | (joy_raw_data[2] << 4);
                andi.b          #$f,d1
                lsl             #4,d1
                move.b          6(a1),d2
                andi.b          #$f,d2
                or.b            d2,d1
                move.b          d1,(a0)+

                move.b          3(a1),d1        *     pack_data[1] = joy_ra
w_data[7] | (joy_raw_data[3] << 4);
                andi.b          #$f,d1
                lsl             #4,d1
                move.b          7(a1),d2
                andi.b          #$f,d2
                or.b            d2,d1
                move.b          d1,(a0)+
```

```
		move.b		4(a1),d1		*	pack_data[2] = joy_ra
w_data[8] | (joy_raw_data[4] << 4);
		andi.b		#$f,d1
		lsl		#4,d1
		move.b		8(a1),d2
		andi.b		#$f,d2
		or.b		d2,d1
		move.b		d1,(a0)+

		move.b		5(a1),d1		*	pack_data[3] = joy_ra
w_data[9] | (joy_raw_data[5] << 4);
		andi.b		#$f,d1
		lsl		#4,d1
		move.b		9(a1),d2
		andi.b		#$f,d2
		or.b		d2,d1
		move.b		d1,(a0)+

		move.b		0(a1),d1		*	pack_data[4] = joy_ra
w_data[1] | (joy_raw_data[0] << 4);
		andi.b		#$f,d1
		lsl		#4,d1
		move.b		1(a1),d2
		andi.b		#$f,d2
		or.b		d2,d1
		move.b		d1,(a0)+

		move.b		11(a1),d1		*	pack_data[5] = joy_ra
w_data[10] | (joy_raw_data[11] << 4);
		andi.b		#$f,d1
		lsl		#4,d1
		move.b		10(a1),d2
		andi.b		#$f,d2
		or.b		d2,d1
		move.b		d1,(a0)+

		rts					* }

		.even
```

```
AS_TMP_BUF:
                .ds.b           12
AS_ACK_BUF:
                .ds.b           5
                .even
AS_RETVAL:
                .dc.w           0
                .dc.l           0
                .dc.l           0
                .end
```

COLUMN

サイバースティックのデータ転送

サイバースティックのデータ転送のやり方について触れておくことにしましょう。ホストから出力するのはREQ 0のみで，サイバースティック側ではこれをスタート信号として他の6本を使ってホストにデータを送るようになっています。まっとうな伝送なら互いに相手が受け取ったか否かをチェックしながらやるのですが，サイバースティックではREQを受け付けた後はスティック側からデータを一方的に垂れ流す方法をとっています。

LHL 0は，データが偶数番目のデータなのか，奇数番目のデータなのかを示すもので，XACKはDAT 01～DAT 31にデータが設定されている(したがって，ホストはデータを読まなくてはならない）ことを示す信号です。

送ってくるデータのフォーマットは図11のようになっています。サイバースティックの仕様書では1回目のA+A′, B+B′のところが単にA，Bとなっていますが，実際はA+A′，B+B′になっています。F，Gは仕様書上はこの呼び方ですが，サイバースティックではSTARTとSELECTスイッチにしています。

仕様書では転送回数が11回となっているのですが，これは間違いのようです。当初，私もこのハード仕様書を信じて11回で転送をやめるようにしていたのですが，自作の読み込みプログラムではちゃんと動くのに，アフターバーナーはまったく動いてくれず，ずいぶん悩みました。シンクロを使ってサイバースティックのタイミングをもう一度全部チェックしていって，やっと気がついたという次第です。そのつもりでドライバのリストを読んでみると，確かに12回こないとエラーとして処理されてしまっています。たぶん，アフターバーナーもこれと同じようにしているために私の回路では動いてくれなかったのでしょう。回路を変更して12回送るようにした（実はこのほうが簡単にできる）ところ，まともに動くようになりました。

最後の12回目に何を渡しているのかは不明です。ドライバのリストを読んでも，ただ読み出しているだけでまったく使用していません。将来の拡張を考えたのかもしれません。

万能リモコン

さまざまな機器に搭載されている赤外線リモコンの波形を記憶，再生できる万能リモコンをX68000で実現してみました。複数の機器を連動させたり本体のタイマ機能と組み合わせたりと，いろいろな応用が考えられることでしょう。

さまざまな機械に赤外線リモコンが使われるようになり，気がついたら部屋の中はリモコンだらけです。特に配線が入り組みやすいAV機器が増えてくると，組み上げた本人ですら，とっさにどのリモコンを使ったらよいのか判断できなくなったりします。

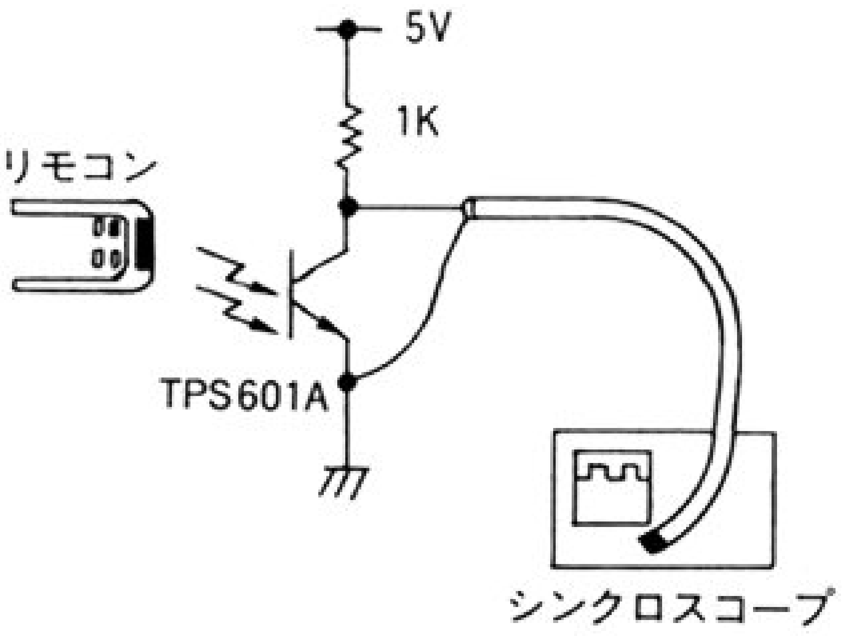
このような状態を少しでも改善しようと，各社のリモコン波形を発生できるようなものや，リモコン波形を記憶させることができるようなものも売られています。これと同じようなことがX68000でもできないでしょうか。X68000で波形を記憶しておいてそれを組み合わせて出力できれば，かなりおもしろいこともできるのではないでしょうか。

1 赤外線リモコンの実験

まず，赤外線リモコンがどのような方法でデータの受け渡しをしているのかを知らなくてはなりません。フォトトランジスタを使った簡単な受光器を作ってリモコンの出力波形を調べてみることにしました。フォトトランジスタには東芝のTPS601Aを使いました。デバイスの上部に凸レンズが付いており指向性が強くなっていますので，外部の影響が少なくなります。試

作した受光器の回路図は図1のようなものです。フォト・トランジスタのシンボルは通常のトランジスタのベースが光になったような格好で，この形のとおり，光があたると，コレクタからエミッタに電流が流れるようになります。

●図……1　試作した受光器の回路図



フォトトランジスタのコレクタにつなぐ抵抗の値の決め方は，少し気を遣う必要があります。抵抗値を大きくしたほうがわずかな光でも0Vまで引ききれるため，感度が上がるのですが，逆に周波数の高い入力信号への応答はだんだん悪くなっていき，大きくしすぎると暗電流（入力がないときに流れる電流）の影響が無視できなくなってきます。抵抗を小さくしていけば，これと逆になります。抵抗をあまり小さくすると，フォト・トランジスタが電流を引ききれなくなったり，最大コレクタ損失を超えて壊れてしまう可能性もありますから極端に小さな値にすることはできません。

データブックを見るとTPS601Aの最大コレクタ損失は150 mWです。たとえば電流が5Vで，抵抗を0 Ωにしてしまったら30 mA(150 mW/5 V)以上は流してはいけないことになります。フォト・トランジスタにあまり負荷をかけたくないので，最大定格を見て抵抗値の限界だけでもつかんでおきます。最大コレクタ電流(IC)は50 mAと，結構流せることがわかります。コレクタ―エミッタ間飽和電圧（VCE(sat)）が0.5Vくらいとして，電源が5Vなら(5−0.5)/50＝0.09(KΩ)だから，90Ωくらいが電流側から計算した限界ということになります。電力から計算した限界のほうは宿題にしておきましょう。コレクタ電流に応じてコレクタ電圧が下がることを考慮するだけですから，さして難しくはありません。ここではかなり余裕をみて1KΩで実験してみることにしました。

❶·1　測　定

とりあえず，リモコン波形のサンプルはCZ-600DEのリモコンからとることにしてみます。

出力をシンクロスコープで観察しながらリモコンのスイッチを押して，距離を調整していきます。当初，RS-232Cのような，1/0を赤外線のON/OFFにした波形を予想していたのですが，実際の波形はかなり異なっています。よく見てみると33 KHz程度の周波数の断続にしているらしいことがわかりました。

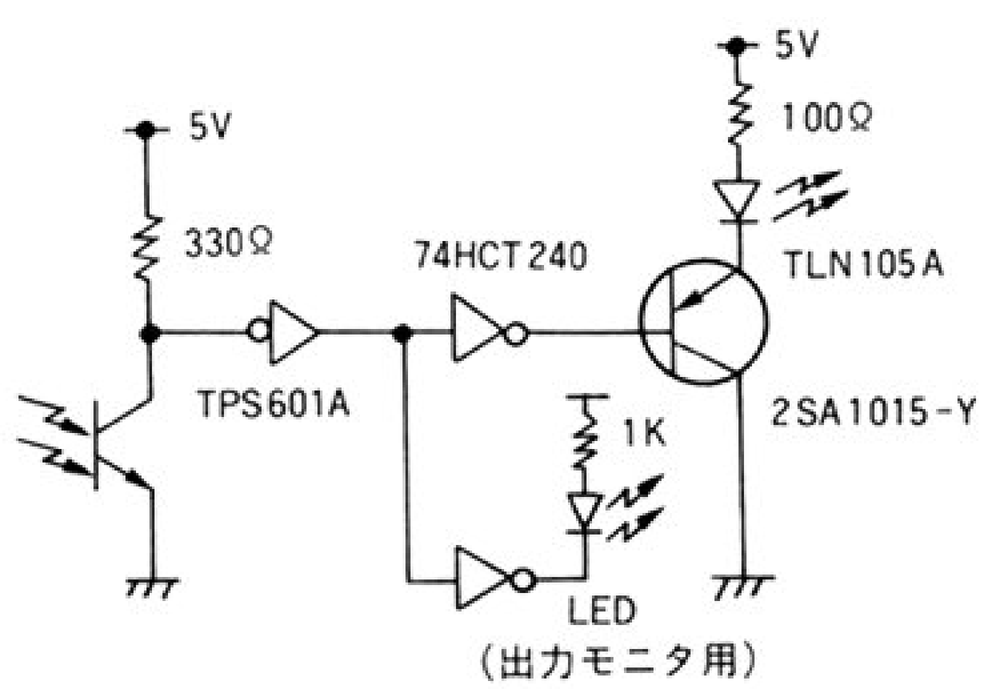
確かに，このようにしておいたほうが外乱に対して強くなります。試みにまったく違うメーカのビデオのリモコンを調べてみても，ほぼ33 KHzの点滅をベースにして，そのON/OFFをやっているという点はまったく同じです。気になって雑誌のパーツショップの広告でリモコン用のモジュールを探してみたら，小さく38 KHzと書いてあります。どうやらこのくらいの周波数をベースにしておくのが一般的な方法のようです。

波形を見る限り，まだ抵抗値が大きいようで，ちょっと近づけるだけでパルスが見えなくなってしまいますので，本番の受光部では抵抗を330 Ωに変更しました。

❶·2 発光側の実験

受け取った波形と同じものをそのまま出力してやれば，専用リモコンと同じようにコントロールできるはずですが，若干不安が残るので，試しに出力段を作ってリピータ（中継器）を作ってみることにしました。図2が実験したリピータの回路です。赤外線発光ダイオードにはTLN105Aを使用しました。TLN105 Aの順電流の最大定格は100 mAということなので，余裕を見込んで30 mAくらい流すことにします。このときのダイオードの両極端の電圧は，IF-VF特性からおよそ1.26 Vくらい。トランジスタのエミッタ電圧がトランジスタのVBE(sat)-IC特性から，およそ0.75 Vくらいとみると，抵抗値は(5 V−1.26 V−0.75 V)/30 mA=0.0997(KΩ)。ということで100 Ωをつけてみました。トランジスタのほうは最大コレクタ電流は150 mA，コレクタ損失も400 mAと十分大きいので計算して確認するまでもないでしょう。

●図……2　リモコンリピータ



できたリピータの赤外線発光ダイオードをディスプレイに，フォトトランジスタにリモコンを向けます。リモコンの右上の赤い電源スイッチを ON。「チッ，ピューン」と聞き慣れた音とともに電源が入りました。チャンネル切り替えもきちんと反応します。リピータの電源を落とすと動かなくなるので，リモコンの直接波ではないことは確実です。

❶·3 本体とのインタフェース

リピータとして動作したのですから，受光部と発光部の間に X68000 がはさまれるようにすればいいわけです。本体との接点にはジョイスティックコネクタを使ってみることにしました。コネクタは，ちょっと隙間が大きくなりますが，普通の 9 ピン D-SUB コネクタが使えます。

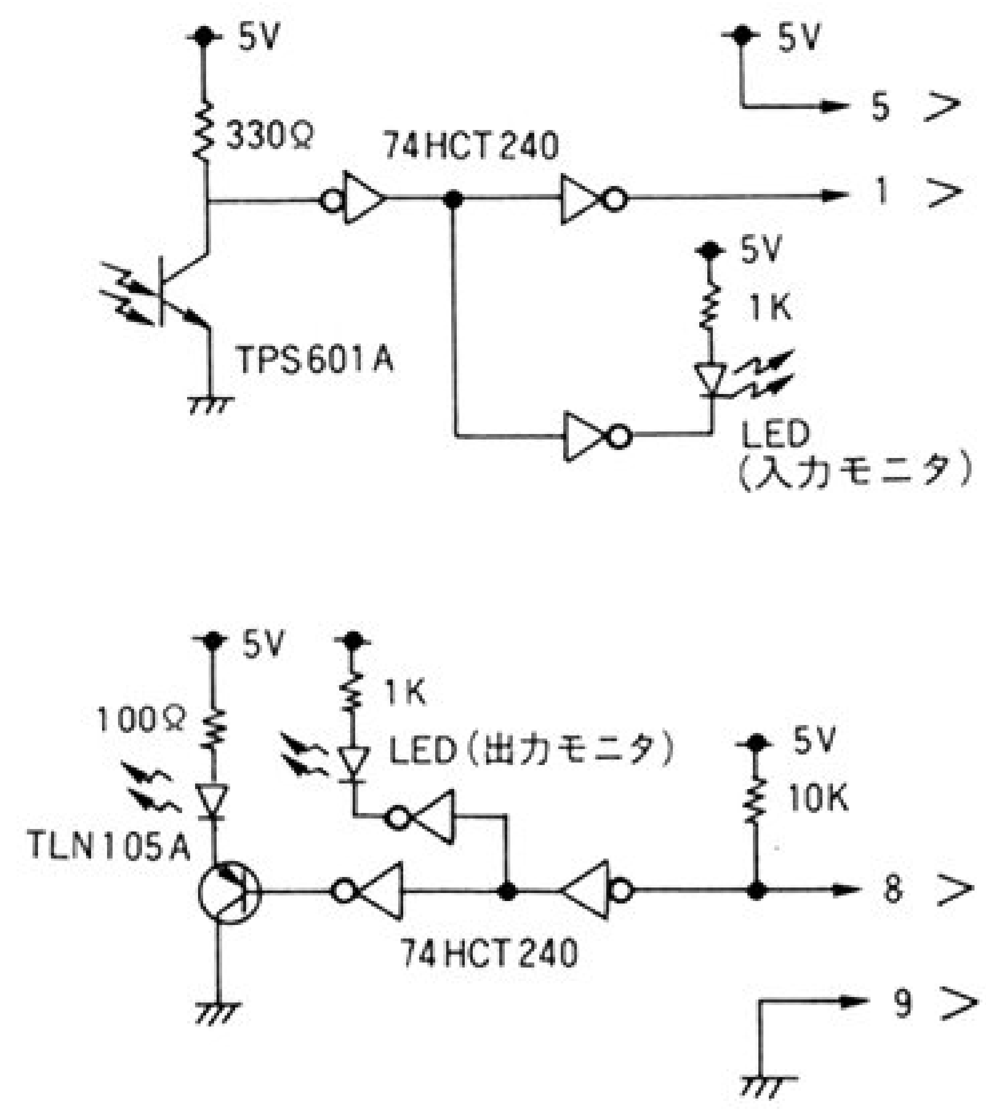
次に，ジョイスティックポートを調べ，どのピンを出力にするかを決めておきましょう。ジョイスティック 1 のポートでは入力専用ピンは 1， 2， 3， 4 番で， 6， 7 番は入出力兼用，8 番は出力専用。電源は 5 番が＋5 V， 9 番が GND となっています。どのピンを使っても似たようなものですが，入力は CPU から見たときにデータの最下位ビットになることから考えて 1 番ピンを，出力は出力専用ピンである 8 番ピンを使うことにしてみました。

❶·4 X68000との接続

33 KHz というのは，そこそこ早い信号なので，取り込みプログラムには気をつけなくてはなりません。33 KHz 以上でデータを取り込むためには，少なくともその 2 倍，すなわち 66 KHz 以上でデータを取り込む必要があります。1/66 KHz＝0.015 mS，つまり 15 μS 以下の間隔で読み取らなくてはなりません。10 MHz の 68000 ではメモリ―メモリ間 MOVE 命令だけで 12 サイクル，DBRA でループを作って 10 サイクルの計 22 サイクル，2.2 μS かかることになります。余裕はあるというものの，あまり余計なことはできません。割り込みが入ったら，15 μS 以内に帰ってくることなど期待できませんから，読み始めたら割り込み禁止で突っ走ることが必要です。

とりあえず，ソフト，ハードともメドが立ったので実際にジョイスティックポートを使ってつないでみることにします。図 3 のような回路を作ってみました。8 番ピンを 10 KΩ の抵抗で 5 V とつないでいるのは，X68000 のジョイスティックインタフェースに使っている i8255 がリセット後はすべて入力ピンになるため，ラインがハイ・インピーダンスになるのが気持ちが悪かったからです。この状態では LED は消灯し，i8255 の初期化が終わると，出力が 0 になるので点灯することになります。

●図……3　赤外線リモコンインタフェース（実験版）



これでまず実験してみました。外部関数を作り，配列を読み込んだ後，連続出力するようにしておきます。リモコンの電源スイッチを押したときの波形を読み込ませたところで，いったん止めて，それから連続出力にします。この状態で CZ-600 DE に向けるとちゃんと電源が入り，向きを変えてからまた向けると，OFF になります。どうにかまともに動いているようです。

❶·5　路離を伸ばす

動作はするようになったのですが，ちょっと気になってどのくらい届くものか実験してみました。ところが結果が芳しくなく，せいぜい 1 m 程度が限界です。普通，赤外線リモコンは 5 m くらい離れても余裕で動きますから，このままでは問題外の低性能です。距離を伸ばすには赤外線をレンズで絞るか，LED に流す電流を増やして光出力を増やすしかありませんが，レンズで絞るのは真正面に向けなくてはならず，使い勝手が悪くなりすぎますので電流の増加を検討するよりありません。しかし，LED に流している電流はすでに最大定格の 3 分の 1 に達しています。定格ぎりぎりまで使っても，たいした改善は期待できそうにありません。

頭を抱えながら再びデータブックを読み直していたら，最大定格に「パルス順電流 IFP」という項目があるのに気がつきました。こちらは 1 A までよいことになっています。この値には

注意書きがしてあって，パルス幅が 100 μS，繰り返し周波数が 100 Hz とあります。わかりにくい書き方ですが，要するに 100 μS の幅で，1 秒間に 100 回，パルス的に点滅させるなら，1 A までは流せるということです。リモコン波形は 33 KHz 程度のパルスで 1 回分のスイッチのデータを送った後，次にデータを送るまでずいぶん時間を取っていますので，この規定を十分満足できそうです。

●2 回路設計

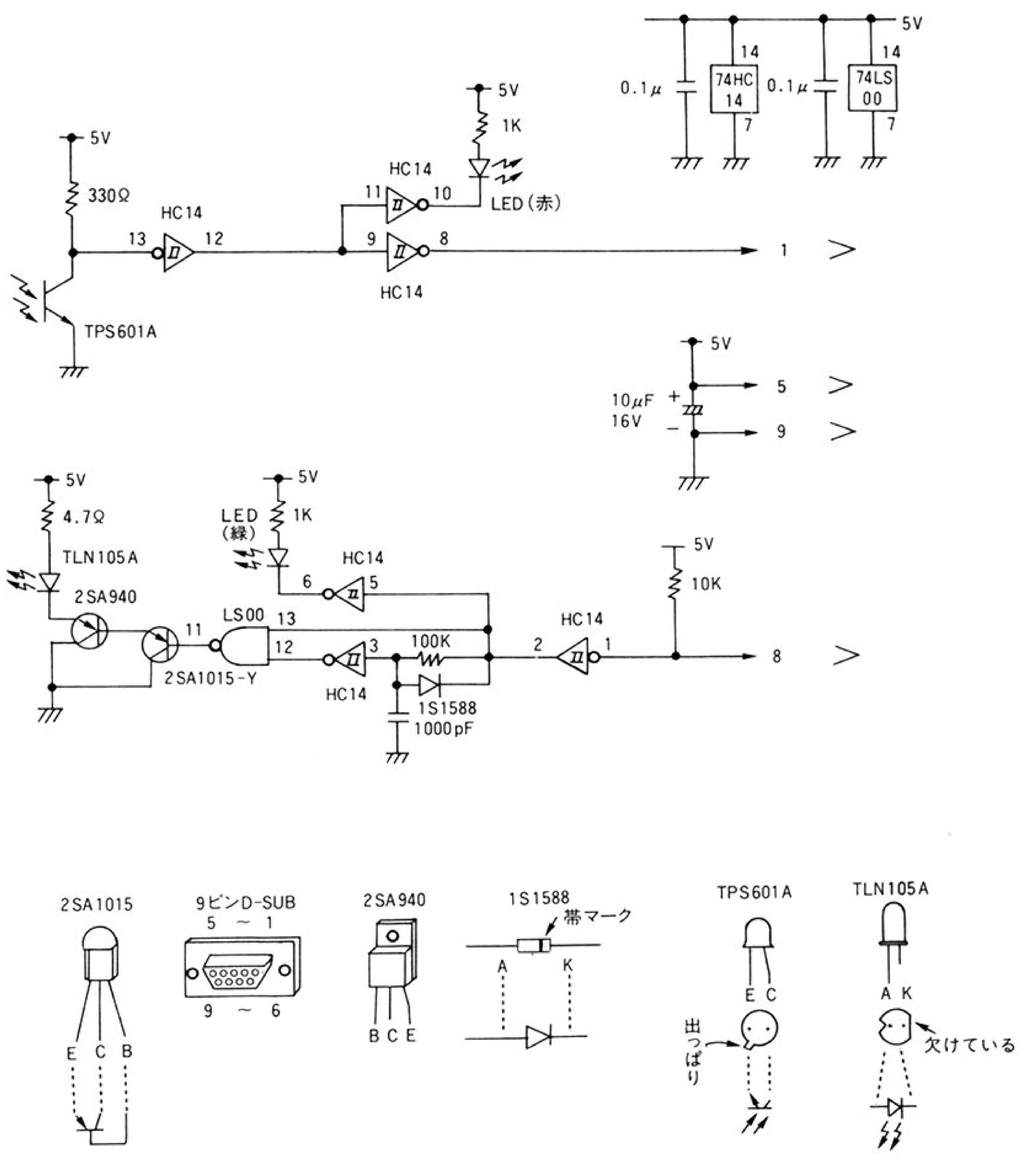
最終的に決定したのが，図 4 の回路図です。図 4 ではインバータを 74HCT240 から 74HC 14 に変更しました。発光側でＣＲのタイミング回路を作るために C-MOS のシュミットゲートが必要だったからです。受光部は回路的には変更なし。送信部はなにやら怪しげな抵抗，コンデンサ，ダイオードが追加され，最終段がダーリントンになり，赤外線発光ダイオードにつながっていた抵抗が 4.7 Ω と一気に小さくなっています。これらについて簡単に説明しておきましょう。

❷·1 連続点灯防止回路

怪しげな抵抗，コンデンサ，ダイオードのあたりは，連続して 100 μS 以上連続発光しないようにするための付加回路です。基本的には 8 番ピンからの入力が 0 になると点灯するのは図 3 と同じですが，入力を受けた 74HC14 の出力からは 100 KΩ の抵抗を通して 1000pF のコンデンサに対して充電が行われます。このコンデンサの電圧が次の 74HC14 のスレッショルド電圧を超えると，この 74HC14 の出力が反転して 0 になり，トランジスタが OFF，したがって赤外線発光ダイオードは消灯するというわけです。この時間はほぼＣとＲの積，すなわち 100×10^{3} $(K\Omega)\times1000\times10^{-12}=100\times10^{-6}$ （秒），100 μS となります。

このままでは，入力が 1 になったときにも 100 μS たたないと，74LS00 につながれた 74HC 14 の出力は 0 に戻りませんから，100 μS 以内に入力が再度 0 になっても，TLN105A は点灯してくれません。このため，入力が 1 になったときにはコンデンサの電荷をすばやく引き抜くよう，抵抗と並列にダイオード（1S1588）をつけておきました。入力が 1 になったときは，このダイオード経由でコンデンサを放電させるわけです。

●図4　赤外線リモコン・インタフェース（最終版）



❷・2　発光ダイオードドライブ回路

TLN105A のドライブはトランジスタをダーリントン接続して行っています。発光ダイオードに流す電流を多くするため，2SA1015 が1つでは間に合わなくなったためです。TLN105A

には300 mA程度流すことにします。このときダイオードの両端は特性図から約1.8 V，2SA940のベース—エミッタ間が約0.6 Vとなります。74LS00のLレベル出力が0.25 V程度とすれば，抵抗値は(5 V−1.8 V−0.7 V−0.6 V−0.25 V)/300 mA＝0.0055 KΩ，5.5 Ωとなります。手持ちのジャンクをあさったら4.7 Ωがあったので，それをつけておきます。連続的に300 mAも流したままにするとしたら，0.3 A×0.3 A×4.7 Ω＝0.423 Wですから最低0.5 W，できたら1 Wの抵抗を使いたいところですが，今回は流す電流がパルスであり，しかも間欠動作ですから，デューティ比は30 %にも満ちません。一般的な1/4 Wの抵抗で十分です。実際に使ってみても，まったく熱くなりません。

❷·3 再実験

この回路で再挑戦してみることにしました。読み込ませたデータを連続出力させながら向きをあわせようとしたら，TLN105Aはとんでもない方向を向いているのに，テレビが突然応答しました。X68000を持ってうろつくわけにはいかないので，距離の測定はできませんでしたが，本物のリモコンにまったくひけをとらないような感触です。

●3 製作上の注意点

工作は各自の流儀でやればよいでしょう。せいぜい100 KHz程度のデジタル信号ですからほとんど気を遣うところはありません。私の作ったものでは部品の使い回しをけっこうするので，ICやフォト・トランジスタ，赤外線発光ダイオードはソケットを使っていますが，まともに買うとICよりもソケットのほうが高いという事態になります。安く作るにはじか付けにしてもかまいません。半田ごての熱を気にする人もいますが，悪意でやらない限り，ICを熱で壊すことはないでしょう（試しに悪意をもってやってみましたが，壊れてくれませんでした)。

❸·1 LED

LEDは単なるモニタ用なので，好きなものを選んでつけてかまいません。私は赤と緑を使い

ました。同じくらいの明るさで点灯させるには緑のほうに少々多めの電流を流さなくてはならないのですが，HC14 の IOL は 4 mA しかないので，赤，緑とも同じ 1 KΩ にしておきました。LS00 のゲートが余っているので，そちらを使えば，もう少し抵抗値を減らしても大丈夫です。

❸·2 ケース

ケースはスチロールケースを使いました。秋葉原の秋月電子通商で 1 個で 90 円で見つけたものです。ケースに穴を開けて基板に固定すると，裏のネジが飛び出すことになり，持ったときの感触がよくないので，ナットをケースに瞬間接着剤で貼り付け，その上にもう 1 つナットを重ねて取り付けてみました。

X68000 本体との接続ケーブルを通すのには穴を開けざるを得ません。きれいな穴にするなら，ハンドドリルなどでゆっくり開けていけばいいのでしょうが，少しきつめの扁平な穴を開けて，ケーブルの固定も兼ねてしまおうと思い，クリップを伸ばしたものをガスレンジで真っ赤になるまで焼いて突き刺して穴を開け，ぐりぐりと何度か動かすという作業を 3 回ほど繰り返して開けてみました。キリや小さいドライバを焼く人もいるようですが，工具を焼きなましすることになりますし，溶けたスチロールが工具にこびりつくので，やらないほうがよいでしょう。

●4 サンプルプログラム

単純なわりにはけっこう楽しめそうなハードウェアなので，ソフトのほうも読み取った波形の修正くらいはできるようなものを作ってみました。データの読み書きのところだけは，先ほどの計算のように時間的な余裕があまりないので，アセンブラで組んであります。ソースは BASIC 外部変数として使うときも，コンパイルしたときにリンクするモジュールにするにもオプションの指定や修正はいっさい必要ないようにしてあります（章末参照）。

読み取ったデータを処理する部分はすべて X-BASIC で書きました。インタプリタでも動かないことはありませんが，波形の縮小表示などは相当時間がかかるので，できるだけコンパイルしてから使ってください。

このプログラムでは波形表示用の表示バッファとデータのコピーやファイル入出力に使うコピーバッファの2つがあります。リモコンの読み取りは，読み取った波形をその場で見て読み取りがうまくいったかどうかを確認したいので，表示バッファに入りますが，その他の入出力はすべてコピーバッファに対して行われます。

それでは，プログラムを走らせてみましょう。

❹·1 セレクトモード

走らせると，まず8×8＝64個の長方形が出てきます。この画面をセレクトモードとかりに呼んでおくことにします。最初は緑の四角ばかりです。カレントディレクトリに信号データがファイルとして存在すれば，それにあたる四角が白枠になり，ファイルのハンドルネームとして登録した名前（ch 1とか）が出てきます。

画面右の修正液（のつもり）アイコンのところにマウスを持っていって左ボタンを押すと，修正液のふたが開きます。ここで左を押したまま右のボタンを押すと，エディットモードに入ります。

❹·2 エディットモード

こうしてエディットモードに入ると，画面左の波形表示のところに表示バッファの内容が示されます。読み取られて表示バッファに入った波形がここに表示されます。一見，30チャンネルのロジックアナライザのような画面ですが，元のデータは1つだけです。表示バッファの内容を任意の場所から任意の拡大/縮小率で見られるようにした結果，このようになりました。ちなみに，表示バッファは32768クロック分です。

画面右のアイコン群は上から順に，リモコン波形の読み取り（目），リモコン波形の出力（豆電球），波形の再表示（方形波），ファイルの書き込み（ファイル），ファイルの読み込み（本），そしてセレクトモードへの復帰（右下のドア）です。マウスカーソルをあわせて左ボタンを押すことで実行されます。ただし，復帰だけは安全のため左ボタンを押した状態で，さらに右ボタンを押さないと実行されないようにしました。

波形の横の小さなものは，スクロールと拡大/縮小表示のアイコンです。その隣の数字が波形の左端がバッファ上のどこになっているかと，拡大/縮小率を示しています。Xが拡大，/が縮小を示します。クリックしながら数字の変化と波形表示の変化を見てもらえれば簡単に理解できると思います。

さて，波形表示のところでマウスの左ボタンを押すと，選択した波形に下線が引かれ，カーソルが現れます(Oカーソルと呼ぶことにします)。そのままマウスを移動すると，もう1つのカーソル（Xカーソル）がマウスについて動き回ります。そこで右ボタンを押します。すると，画面の一番下にNULLからSET，RESET，COPY，PASTE，JUMPとモードの切り替わりが表示されます。

NULLはなにもしない。SETはOカーソルからXカーソルまでの間をLEDの点灯状態(波形が上側にある状態）にします。RESETは，2つのカーソルの間をLEDの消灯状態（同じく下側にある状態)にします。COPYは，その区間のデータをコピーバッファに転送します。PASTE(画面ではPASTになってしまっている。あしからず)は，コピーバッファの内容をOカーソルの位置からデータバッファにコピーします。JUMPは，Oカーソルの位置を画面の左端になるように移動するものです。

❹·3 使ってみる

使い方としては，まずデータを読み取らせ，画面に表示された波形をSET，RESETで整形し，必要な部分をCOPYで取り込み，試しに出力アイコンで出力してみて，うまく相手の機器が動くようなら，バッファの内容をファイルに落とすという流れになるでしょう。そこで，リモコンから信号を読み取ることから始めてみましょう。巷に出回っているインテリジェントリモコンにデータを入れる要領です。

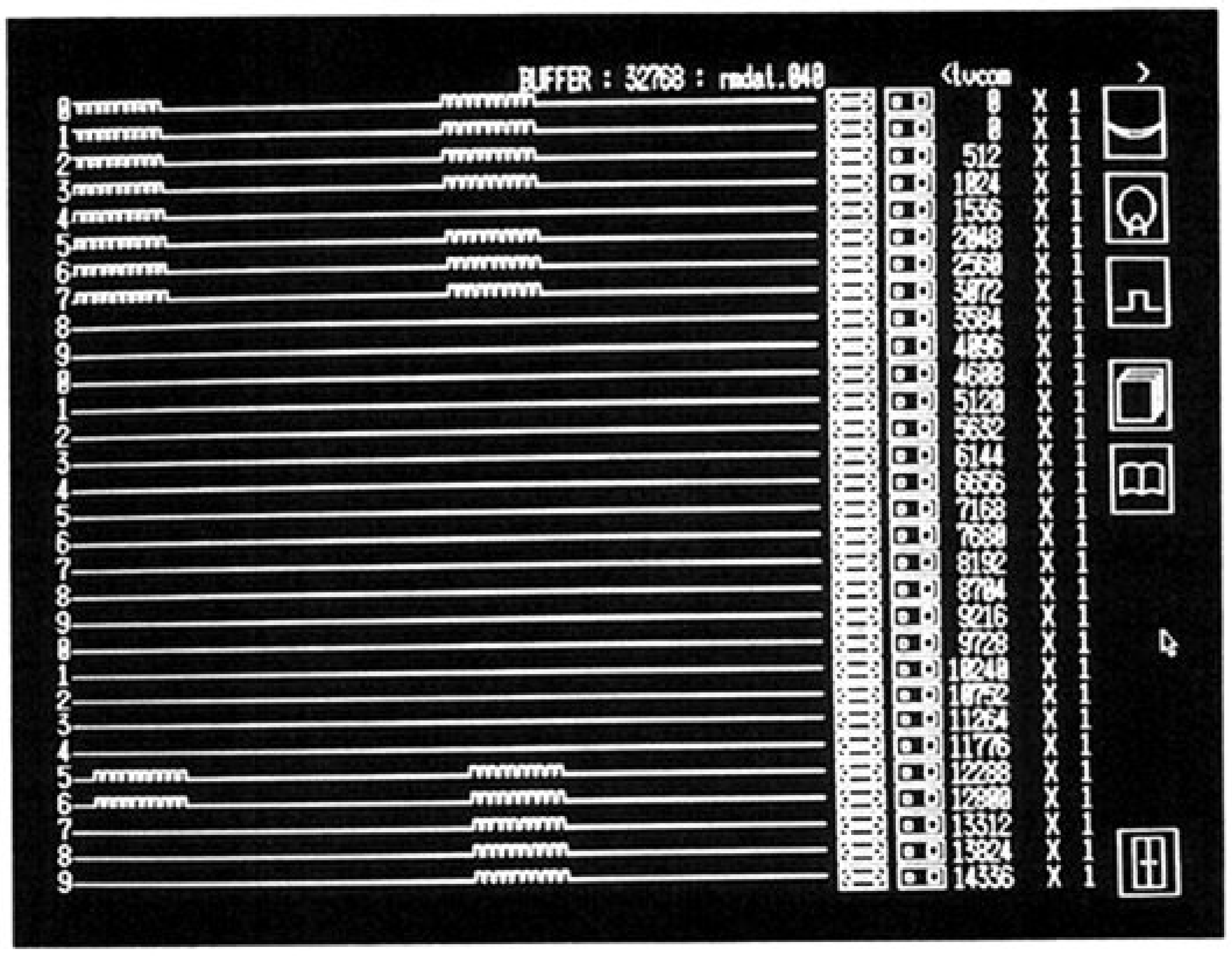
まず，エディットモードに入って読み取りアイコンをクリックすると，アイコンの閉じていた目が開きます。これが読み取り待機の状態です。製作したリモコンを既製のリモコンと向かい合わせて信号を読み取らせます。受光部のLEDが点滅するのが，信号が入ったしるしで，画面には読み取られた波形が表示されます（写真1）。

次に，この波形をコピーバッファに取り込みます。前述したやり方でOカーソルとXカーソルで範囲を決め，COPYを実行するわけです。読み取った波形データ全体をCOPYするには，拡大/縮小アイコンを使って波形表示すればよいでしょう。ここで出力アイコンを使って波形を出してみれば相手の機器が動作するかどうかを確認できます。

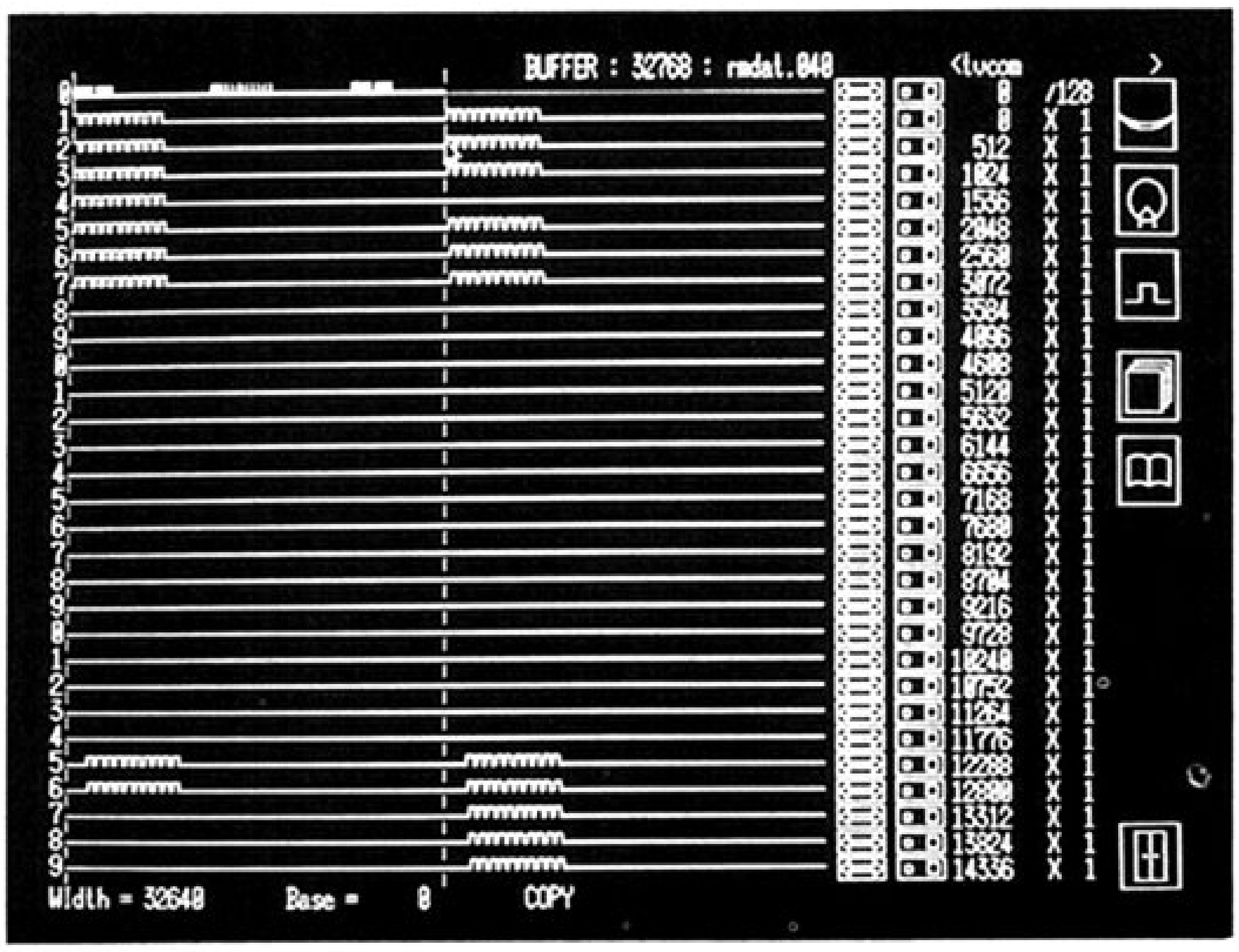
たとえば，テレビのリモコンからチャンネルコールの信号を読み取った場合，まず製作したリモコンをテレビに向けてから出力アイコンをクリックします。そのとき，テレビ画面にチャンネル表示がされれば，きちんと働いているわけです（もし機能していないようだったら，波形の読み取りをやり直してみましょう）。バッファに正しい波形が入っていることを確認できたら，次はそれを登録します（写真2）。

書き込みアイコンをクリックすると，ファイル名に続いてハンドルネームを聞いてきます。

●写真……1　エディットモード

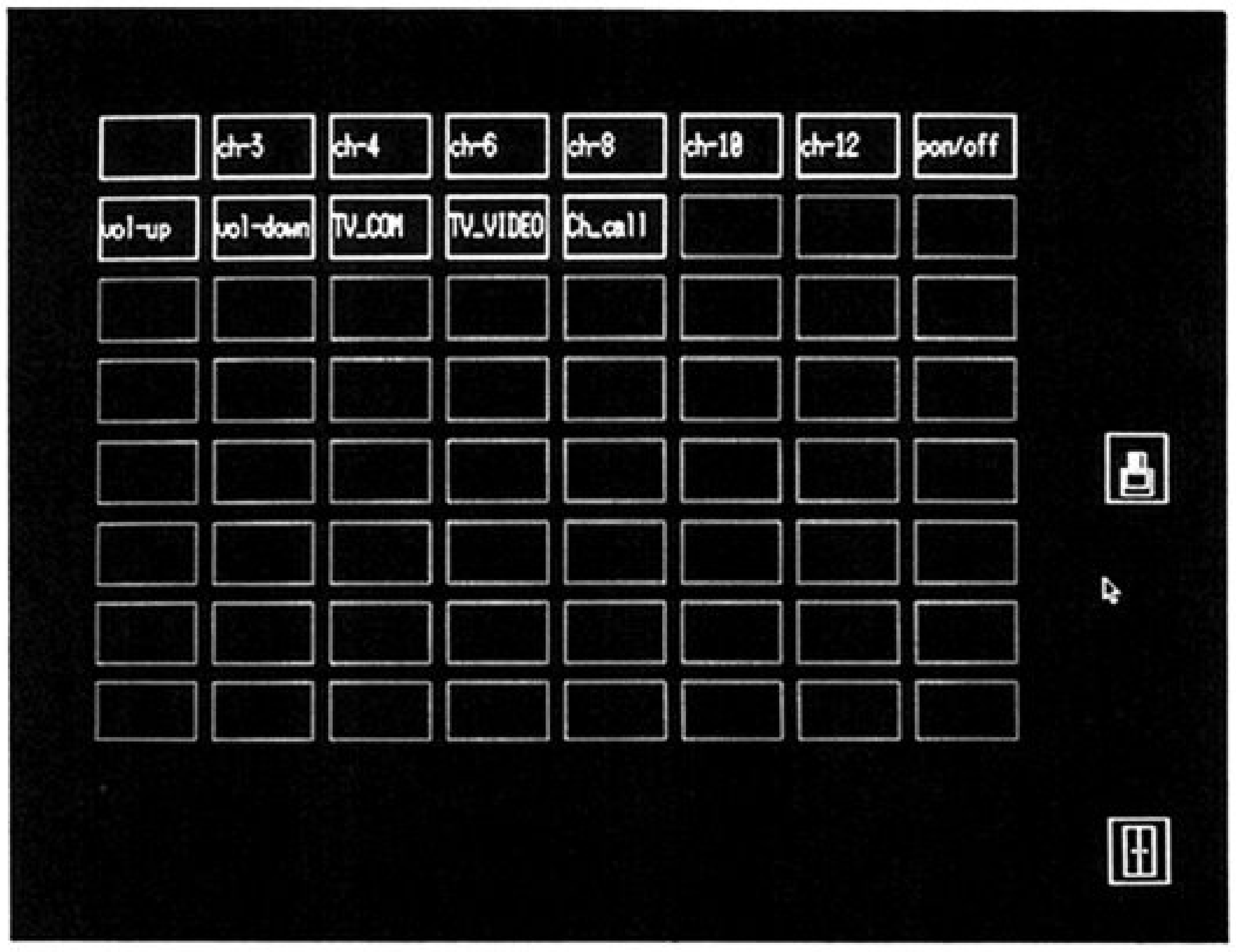


●写真……2　コピーバッファに読み取った波形が表示される



このハンドルネームは，セレクタモードに出てくる四角の中に表示される文字データになります。その波形データの機能名（チャンネル番号など）を入れておくとよいでしょう。ファイル名のほうは固定で，セレクトモードの左上に表示される長方形が rmdat.000，以下横に rmdat.001，rmdat.002 と進んで，最後は右下の rmdat.063 にしています。全部で 64 個のファイルが登録できることは先述のとおりです。書き込みが終わったら，セレクトモードの長

●写真……3　セレクトモード



方形がちゃんとハンドルネーム付きで白くなっていることを確認してください（写真3）。

こうして登録したデータをセレクトモードで選択する（白い枠になっている長方形をクリックする）と，ディスクからデータを読み込み，リモコン出力が行われます。このときも，そのデータはCOPYバッファに入ります。セレクトモードで動作させてみて，具合が悪ければ，そのままエディットモードに入ればよいわけです。

SET，RESETやCOPYでデータバッファの内容が変化すると，そのラインの横の番号だけが白という状態になります。これは，他のラインの表示データがすでに古いものであることを示しています。これが書き直されるのは，その波形がセレクトモードで選択され直したときか，あるいは右の再表示アイコンがクリックされ，すべての波形の書き直しが指示されたときです。変更があるたびにすべてを書き直さないのは，なにかするたびにすべてを書き直すのを待たせず，応答をよくしたいということと，過去のデータを下にリファレンスとして置いておこうという考えからです。

エディットモードでの作業が終了したら，右下のドアのアイコンのところで左ボタンを押すと，ドアが開きます。そのまま右のボタンを押すと，セレクトモードの画面に戻ります。ダブルクリックも考えたのですが，右のボタンが遊んでいるのがもったいない感じだったので，このようにしました。

プログラムを終了するときはセレクトモードの画面の右下のドアで，先ほどと同じように左一右の順でクリックすればよいようになっています。

●5 おわりに

ワイヤレスリモコンは作ってみるとかなり楽しめるものでした。リモコン付きの機器であれば何でもX68000から制御できるようになるわけです。自分で作ったオリジナル機器のコントロールに使うのもおもしろいでしょう。AV機器の波形を取り込んでおいて，各機器を連動させるようにすれば，まさにインテリジェントリモコンと呼ぶにふさわしいこともできるのではないでしょうか。

●リスト……1　赤外線コントローラサンプルプログラム

```
/****************************************
/*                                      *
/* 赤外線コントローラサンプルプログラム *
/*                                      *
/*   1989-02-24 Programmed by M.kuwano  *
/*        -03-05 Remove Slight Bugs     *
/*                                      *
/****************************************
char c(32768),buf(32768),filebuf(4096)
char mv(256*4),zm(256)
int start(30),zosq(30),redraw_flag(30)
int exist_flag(64)
int s,fp,mode,bufsize,fbufsize,fsize,exitflag
dim str modes(5) = {"NULL ","SET  ","RESET","COPY ","PAST ","JUMP "}
dim bitmsk(7)    = {1,2,4,8,&H10,&H20,&H40,&H80}
str fname,hname
screen 2,0,1,1
console 0,32,0
bufsize = 0
fname = "":hname=""
clear_buf():clear_copybuf()
gen_icon()
mouse(0):mouse(1):mouse(4)
repeat
    exitflag = selector()
```

```
    if exitflag then break
    editor()
until 0
end
func editor()
    int i
    screen 2,0,1,1
    console 0,32,0
    draw_ctrl_icon()
    disp_fname()
    start(0)=0:zosq(0)=1
    draw_icon(0):draw_wave(0)
    for i=1 to 29
        start(i) = (i-1)*512
        zosq(i) = 1
        draw_icon(i)
        draw_wave(i)
    next
    edit_wave()
endfunc
func draw_wave(num;int)
    int i
    fill(10,num*16+16,511+10,num*16+31,0)
    locate 76,num+1:print using "#####   ";start(num);
    i = zosq(num): if i=0 or i=-1 then i = 1
    if i<0 then print using "/###";-i; else print using "X###";i;
    if zosq(num)<0 then i=1 else if zosq(num) > 1 then i=2 else i=0
    switch(i)
       case  0: draw_wave_normal(num):break
       case  1: draw_wave_sqz(num):break
       case  2: draw_wave_zoom(num):break
    endswitch
    redraw_flag(num) = 0
    disp_number(num,15)
endfunc
func draw_wave_normal(num;int)
    int i,pdat,pp,base
    base = num*16+16
    pp = start(num)
    if pp >= 32768 then return(0)
```

```
    pdat = c(pp)
    pset(10,base+5*(pdat and 1),15)
    for i=1 to 511
        pp = pp+1
        if pp >= 32768 then break
        if pdat <> c(pp) then {
            line(i+10,base,i+10,base+5,15)
        } else {
            pset(i+10,base+5*(c(pp) and 1),15)
        }
        pdat=c(pp)
    next
endfunc
func draw_wave_sqz(num;int)
    int pdat,px,pp,np,base,zoom,brk
    base = num*16+16
    pp = start(num)
    if pp >= 32768 then return(0)
    pdat = c(pp)
    px = 0
    zoom = -zosq(num)
    brk = 0
    while brk=0 and px<=511
        np = pp + zoom:if np >= 32768 then np = 32768:zoom = np - pp:brk = 1
        switch chk_edge(pdat,pp,zoom)
            case   1:pset(px+10,base,15):break
            case   2:pset(px+10,base+5,15):break
            case   3:line(px+10,base,px+10,base+5,15):break
        endswitch
        pp = np
        pdat = c(pp-1)
        px = px+1
    endwhile
endfunc
func draw_wave_zoom(num;int)
    int pdat,ndat,px,nx,pp,np,base,zoom,brk
    base = num*16+16
    pp = start(num)
    if pp >= 32768 then return(0)
    pdat = c(pp)
```

```
    px = 0
    zoom = zosq(num)
    brk = 0
    while brk=0
        np = pp+1:if np >= 32768 then break
        ndat = c(np)
        nx   = px+zoom:if nx >= 511 then nx = 511:brk = 1
        line(px+10,base+5*(pdat and 1),nx+10,base+5*(pdat and 1),15)
        if brk then break
        if pdat <> ndat then line(nx+10,base,nx+10,base+5,15)
        pp = np
        pdat = ndat
        px = nx
   endwhile
endfunc
func chk_edge(pastdat;int,pp;int,zoom;int)
   int i
   if (pastdat and 1) = 0 then pastdat = 1 else pastdat = 2
   for i=1 to zoom
       if (c(pp) and 1) = 0 then pastdat = pastdat or 1 else pastdat = pastd
at or 2
       if pastdat = 3 then break
       pp = pp+1
   next
   return(pastdat)
endfunc
func gen_icon()
    int x,y
    x=16:y=16
    box(x,y,x+31,y+13,15)
    line(x+2,y+6,x+7,y+2,15)
    line(x+2,y+7,x+7,y+11,15)
    line(x+7,y+2,x+7,y+5,15)
    line(x+7,y+11,x+7,y+8,15)
    line(x+7,y+5,x+24,y+5,15)
    line(x+7,y+8,x+24,y+8,15)
    line(x+24,y+5,x+24,y+2,15)
    line(x+24,y+8,x+24,y+11,15)
    line(x+24,y+2,x+29,y+6,15)
    line(x+24,y+11,x+29,y+7,15)
```

```
    get(x,y,x+31,y+15,mv)
    wipe()
    box(x,y,x+31,y+13,15)
    box(x+5,y+5,x+9,y+9,15)
    box(x+20,y+3,x+27,y+10,15)
    get(x,y,x+31,y+15,zm)
    wipe()
endfunc
func draw_icon(n;int)
    put(530,n*16+16,530+31,n*16+16+15,mv)
    put(570,n*16+16,570+31,n*16+16+15,zm)
    disp_number(n,15)
endfunc
func edit_wave()
    int x,y,bl,br,mscmd,num,stp,exitflag,bcount
exitflag = 0
bcount = 1000
repeat
    msstat(x,y,bl,br):mspos(x,y)
    if bl <> 0 then {
        if bcount > 0 then bcount = bcount-1
        if bcount > 0 and bcount < 999 then continue
        mscmd =  edw_ms_chk(x,y)
        num = mscmd / 256
        mscmd = mscmd and 255
        switch mscmd
            case    1: edw_movl(num)
                       break
            case    2: edw_movr(num)
                       break
            case    3: edw_sqz(num)
                       break
            case    4: edw_zoom(num)
                       break
            case &H10: edw_cursor(num)
                       break
            case &H20: edw_read()
                       break
            case &H21: edw_write()
                       break
```

```
                case &H22: edw_redraw()
                           break
                case &H23: edw_fwrite()
                           break
                case &H24: edw_fread()
                           break
                case &H25: exitflag = edw_exit()
                           break
                default:   break
            endswitch
        } else bcount = 1000
    until exitflag = 1
endfunc
func edw_ms_chk(x;int,y;int)
    int retdat
    retdat = 0
    if y < 16*31 and y >= 16 then {
        if x >= 530 and x < 602 then {
            if x>=530 and x<546 then retdat=1
            if x>=546 and x<562 then retdat=2
            if x>=570 and x<586 then retdat=3
            if x>=586 and x<602 then retdat=4
            retdat = retdat+(y/16-1)*256
            return(retdat)
        }
        if x < 530 then {
            retdat = (y/16-1)*256+&H10
            return(retdat)
        }
    }
    if x >= 720 and x < 760 then {
        if y >= 16 and y < 55  then retdat = &H20
        if y >= 66 and y < 105 then retdat = &H21
        if y >= 116 and y < 155 then retdat = &H22
        if y >= 176 and y < 215 then retdat = &H23
        if y >= 226 and y < 266 then retdat = &H24
        if y >= 460 and y < 500 then retdat = &H25
    }
    return(retdat)
endfunc
```

```
func edw_movl(num;int)
    int stp
    stp = zosq(num)
    if stp > 0 then stp = 10/stp else stp = -stp*10
    if stp = 0 then stp = 1
    stp=stp+start(num)
    if stp >= 32768 then stp = 32768
    start(num) = stp
    draw_wave(num)
endfunc
func edw_movr(num;int)
    int stp
    stp = zosq(num)
    if stp > 0 then stp = 10/stp else stp = -stp*10
    if stp = 0 then stp = 1
    stp = start(num) - stp
    if stp <= 0 then stp = 0
    start(num) = stp
    draw_wave(num)
endfunc
func edw_sqz(num;int)
    stp = zosq(num):if stp = 0 or stp = 1 then stp = -1
    if stp < 0 then stp = stp*2 else stp = stp/2
    if stp < -999 or stp > 999 then stp = zosq(num)
    zosq(num) = stp
    draw_wave(num)
endfunc
func edw_zoom(num;int)
    stp = zosq(num):if stp = 0 or stp = -1 then stp = 1
    if stp < 0 then stp = stp/2 else stp = stp*2
    if stp < -999 or stp > 999 then stp = zosq(num)
    zosq(num) = stp
    draw_wave(num)
endfunc
func edw_cursor(num;int)
    int px,ox,x,y,pbr,br,bl,mode
    pbr = 0
    mode = 0
    if redraw_flag(num) = 1 then draw_wave(num)
    mspos(px,y)
```

```
    px = set_cursor(num,px)
    ox = px
    line(10,num*16+16+6,521,num*16+16+6,9)
    line(ox,0,ox,495,15,&HAA)
    line(px,0,px,495,15,&H55)
    disp_width(csrtopos(num,px,ox))
    disp_base(csrtobpos(num,ox))
    disp_mode(mode)
    bl = -1
    while bl = -1
        msstat(x,y,bl,br)
        if x <> 0 then {
            mspos(x,y)
            x = set_cursor(num,x)
            line(px,0,px,495,0,&H55)
        line(x,0,x,495,15,&H55)
        px = x
        disp_width(csrtopos(num,x,ox))
    }
    if br <> 0 then {
        if pbr = 0 then {
            mode = (mode + 1) mod 6
            disp_mode(mode)
        }
        pbr = -1
    } else pbr = 0
endwhile
mouse(2)
switch mode
    case    1: edw_set_data(num,px,ox)
               set_redraw_flag()
               draw_wave(num)
               break
    case    2: edw_reset_data(num,px,ox)
               set_redraw_flag()
               draw_wave(num)
               break
    case    3: edw_copy_data(num,px,ox)
               draw_wave(num)
               break
```

```
        case    4: edw_past_data(num,ox)
                   set_redraw_flag()
                   draw_wave(num)
                   break
        case    5: edw_jump(num,ox)
                   draw_wave(num)
                   break
        default:   break
    endswitch
    line(px,0,px,495,0,&HFF)
    line(ox,0,ox,495,0,&HFF)
    line(10,num*16+16+6,521,num*16+16+6,0)
    locate 0,31:print space$(90);
    mouse(1)
endfunc
func set_cursor(num;int,x;int)
    int posdat
    posdat = csrtobpos(num,x)
    if posdat > 32768 then {
        x = postocsr(num,32768)
    } else x = postocsr(num,posdat)
    return(x)
endfunc
func edw_set_data(num;int,x;int,ox;int)
    int i,st,ed
    st = csrtobpos(num,ox)
    ed = csrtobpos(num,x)
    if ed < st then i=st:st=ed:ed=i
    ed = ed-1
    for i=st to ed
        c(i)=8
    next
endfunc
func edw_reset_data(num;int,x;int,ox;int)
    int i,st,ed
    st = csrtobpos(num,ox)
    ed = csrtobpos(num,x)
    if ed < st then i=st:st=ed:ed=i
    ed = ed-1
    for i=st to ed
```

```
        c(i)=9
    next
endfunc
func edw_copy_data(num;int,x;int,ox;int)
    int i,j,st,zoom
    st = csrtobpos(num,ox)
    ed = csrtobpos(num,x)
    if ed < st then i=st:st=ed:ed=i
    ed = ed-1
    j=0
    for i=st to ed
        buf(j)=c(i)
        j=j+1
    next
    bufsize = j
    disp_fname()
endfunc
func edw_past_data(num;int,ox;int)
    int i,j,st,zoom
    st = csrtobpos(num,ox)
    i = st
    for j=0 to bufsize-1
        if i >= 32768 then break
        c(i)=buf(j)
        i=i+1
    next
endfunc
func edw_jump(num;int,ox;int)
    int st
    st = csrtobpos(num,ox)
    if st >= 32768 then st = 32768
    start(num)=st
endfunc
func edw_read()
    int i
    mouse(2)
    draw_open_eye()
    rmread(32768,c)
    draw_close_eye()
    edw_redraw()
```

```
    mouse(1)
endfunc
func edw_write()
    int x,y,br,bl,i,j,col
    i = 1:col = 15
    if bufsize = 0 then return(0)
    repeat
        i = i - 1
        if i <= 0 then {
            i = 1000/bufsize+1
            draw_light(col)
            if col = 0 then col = 15 else col = 0
        }
        rmwrite(bufsize,buf)
        for j=0 to 3000:next
        msstat(x,y,bl,br)
    until bl = 0
    draw_light(0)
endfunc
func csrtobpos(num;int,x;int)
    return(csrtopos(num,x,10)+start(num))
endfunc
func csrtopos(num;int,px;int,ox;int)
    int zoom
    zoom = zosq(num)
    if zoom = 0 then return(px-ox)
    if zoom < 0 then return((ox-px)*zoom)
    return((px-ox)/zoom)
endfunc
func postocsr(num;int,px;int)
    int cpos,zoom,st
    zoom = zosq(num):if zoom = 0 then zoom = 1
    st = start(num)
    if zoom < 0 then cpos = (st-px)/zoom+10 else cpos = (px-st)*zoom+10
    if cpos < 10 then cpos = 10
    if cpos > 521 then cpos = 521
    return(cpos)
endfunc
func disp_mode(mode;int)
    locate 40,31:print modes(mode mod 6);
```

```
endfunc
func disp_base(b;int)
    locate 20,31:print using "Base = #####";b;
endfunc
func disp_width(w;int)
    if w < 0 then w = -w
    locate 0,31:print using "WIdth = #####";w;
endfunc
func clear_buf()
    int i
    for i=0 to 32767
        c(i)=9
    next
endfunc
func clear_copybuf()
    int i
    for i=0 to 32767
        buf(i)=9
    next
endfunc
func edw_fwrite()
    int i,fp,sz,x,y,br,bl
    str s
    if bufsize = 0 then return(0)
    console 31,1,0
    input"ファイル名は何にしましょうか ";s
    if s = "" then s = fname
    if s <> "" then {
        error off
        fp = fopen(s,"c")
        if fp <> -1 then {
            input"ハンドル・ネームはなににしますか ";s
            if s = "" then s = hname
            s = s+chr$(&HD)+chr$(&HA)
            mouse(2)
            locate 0,31:print"圧縮中です。      ";
            fbufsize = compress()
            locate 0,31:print"書き込み中です。";
            fwrites(s,fp)
            fputc(bufsize/256,fp)
```

```
            fputc(bufsize mod 256,fp)
            sz = fwrite(filebuf,fbufsize,fp)
            fclose(fp)
            mouse(1)
            locate 0,31
            if sz < fbufsize then {
                print"書ききれないんだけど・・";:beep
            } else print"書きおわりました。";
        } else {
            locate 0,31
            print"ファイル名がおかしいような気がするんですけど・・・";
            beep
        }
        error on
        repeat
            msstat(x,y,bl,br)
        until x<>0 or y<>0
    } else beep
    locate 0,31:print space$(80)
    console 0,32,0
endfunc
func edw_fread()
    int i,fp,x,y,br,bl
    str s
    console 31,1,0
    input"何を読みましょうか ";s
    if s = "" then s = fname
    console 0,32,0
    if s <> "" then {
        error off
        fp = fopen(s,"r")
        if fp <> -1 then {
            mouse(2)
            clear_copybuf()
            locate 0,31:print"読み込み中です。";
            freads(hname,fp)
            bufsize = fgetc(fp)*256
            bufsize = bufsize + fgetc(fp)
            fbufsize = fread(filebuf,4096,fp)
            fclose(fp)
```

```
            mouse(1)
            locate 0,31
            print"データ・サイズは";bufsize;"でした。";
            fname = s
            disp_fname()
            printf"・・・データ拡張中です。";
            swell()
        } else {
            locate 0,31
             print"ファイルが無いような気がするんですけど・・・";
            beep
        }
        error on
        repeat
            msstat(x,y,bl,br)
       until x<>0 or y<>0
    }
    locate 0,31:print space$(80);
    console 0,32,0
endfunc
func edw_redraw()
    int i
    for i=0 to 29
        draw_wave(i)
    next
endfunc
func disp_fname()
    locate 40,0
    print space$(55);
    locate 40,0
    print using "BUFFER : ##### : &              & <&                &>";buf
size,fname,hname;
endfunc
func set_redraw_flag()
    int i
    for i=0 to 29
        redraw_flag(i) = 1
        disp_number(i,7)
    next
endfunc
```

```
func disp_number(num;int,col;int)
    symbol(0,num*16+12,chr$(&H30+(num mod 10)),1,1,1,col,0)
endfunc
func edw_exit()
    int x,y,br,bl,retdat
    draw_open_door()
    retdat = 0
    repeat
        msstat(x,y,bl,br)
        if bl<>0 and br<>0 then retdat = 1:break
    until bl = 0
    if retdat = 0 then draw_close_door()
    return(retdat)
endfunc
func compress()
    int i,bit,bufp
    bufp = 0
    bit = 0
    filebuf(0) = 0
    for i = 0 to bufsize-1
        if buf(i) and 1 then filebuf(bufp) = filebuf(bufp) or bitmsk(bit)
        bit = bit + 1:if bit > 7 then bufp = bufp + 1:filebuf(bufp)=0:bit
= 0
    next
    return(bufp+1)
endfunc
func swell()
    int i,bit,bufp
    bufp = 0
    bit = 0
    for i=0 to bufsize-1
        if filebuf(bufp) and bitmsk(bit) then buf(i) = 9 else buf(i) = 8
        bit = bit + 1:if bit > 7 then bufp = bufp + 1:bit = 0
    next
endfunc
func draw_ctrl_icon()
    box(720,16,760,56,15)
    box(720,66,760,106,15)
    box(720,116,760,156,15)
    box(720,176,760,216,15)
```

```
    box(720,226,760,266,15)
    box(720,460,760,500,15)
    draw_midget_lamp()
    draw_light(0)
    draw_close_eye()
    draw_redraw_wave()
    draw_file()
    draw_book()
    draw_close_door()
endfunc
func draw_open_eye()
    fill(721,17,759,55,0)
    circle(740,16,25,15,227,313)
    circle(740,55,25,15,45,135)
    circle(740,36,5,15)
    paint(740,36,3)
endfunc
func draw_close_eye()
    fill(721,17,759,55,0)
    circle(740,16,28,15,230,310)
    circle(740,0,41,15,242,298)
endfunc
func draw_light(col;int)
    line(740,71,740,69,col)
    line(725,86,723,86,col)
    line(755,86,757,86,col)
    line(731,77,728,74,col)
    line(749,77,752,74,col)
    line(731,95,729,97,col)
    line(749,95,751,97,col)
    if col = 0 then paint(740,86,0) else paint(740,86,7)
endfunc
func draw_midget_lamp()
    circle(740,86,12,15)
    line(740,90,735,95,15)
    line(740,90,745,95,15)
    line(735,95,735,102,15)
    line(745,95,745,102,15)
endfunc
func draw_redraw_wave()
```

```
    line(725,146,735,146,15)
    line(735,146,735,136,15)
    line(735,136,745,136,15)
    line(745,136,745,146,15)
    line(745,146,755,146,15)
endfunc
func draw_file()
    box(725,187,749,211,15)
    line(728,187,728,184,15)
    line(728,184,752,184,15)
    line(752,184,752,208,15)
    line(752,208,749,208,15)
    line(731,184,731,181,15)
    line(731,181,755,181,15)
    line(755,181,755,205,15)
    line(755,205,752,205,15)
endfunc
func draw_book()
    circle(733,266,13,15,57,123)
    circle(747,266,13,15,57,123)
    circle(733,250,13,15,57,123)
    circle(747,250,13,15,57,123)
    line(726,255,726,239,15)
    line(754,255,754,239,15)
    line(740,255,740,239,15)
endfunc
func draw_close_door()
    fill(721,461,759,499,0)
    box(730,465,750,495,15)
    line(740,465,740,495,15)
    circle(736,480,1,15)
    circle(744,480,1,15)
endfunc
func draw_open_door()
    fill(721,461,759,499,0)
    line(730,465,750,465,15)
    line(730,465,730,495,15)
    line(750,465,750,495,15)
    line(738,495,742,495,15)
/*
```

```
    line(730,465,738,468,15)
    line(738,470,738,498,15)
    line(730,495,738,498,15)
    line(750,465,742,468,15)
    line(750,495,742,498,15)
    line(742,468,742,498,15)
    circle(735,482,1,15)
    circle(745,482,1,15)
endfunc
/*
/************** セレクター **************/
/*
/* draw_open_door,daw_close_doorは共用です。
/* 切り放して使う時は忘れずに。
/*
func selector()
    int x,y,br,bl,px,py,fp,retdat
    screen 2,0,1,1
    console 0,32,0
    draw_sel_block()
    retdat = 0
    repeat
        msstat(x,y,bl,br)
        if bl<>0 then {
            mspos(x,y)
            if x>=30 and x<658 and y>=30 and y<410 then {
              px = (x-30)/80:py = (y-38)/48
              if x-(px*80+30)<68 and y-(py*48+38)<36 and exist_flag(py*8+p
x) then {
                     paint(px*80+32,py*48+40,5)
                     sel_output(py*8+px)
                     paint(px*80+32,py*48+40,0)
                }
                continue
            }
            if x >= 720 and x < 760 then {
                if y >= 226 and y < 266 then {
                    draw_open_white()
                    repeat
                        msstat(x,y,bl,br)
```

```
                    until (bl = 0) or (bl<>0 and br<>0)
                    if bl<>0 and br<>0 then break
                    draw_close_white()
                    continue
                }
                if y >= 460 and y < 500 then {
                    draw_open_door()
                    repeat
                        msstat(x,y,bl,br)
                    until (bl = 0) or (bl<>0 and br<>0)
                    if bl<>0 and br<>0 then retdat = 1:break
                    draw_close_door()
                    continue
                }
            }
        }
    until 0
    return(retdat)
endfunc
func sel_output(num;int)
    int i,fp,x,y,bl,br
    fname = "rmdat."+right$("00"+str$(num),3)
    fp = fopen(fname,"r")
    if fp=-1 then return(0)
    freads(hname,fp)
    bufsize = fgetc(fp)*256
    bufsize = bufsize + fgetc(fp)
    fbufsize = fread(filebuf,4096,fp)
    fclose(fp)
    swell()
    repeat
        rmwrite(bufsize,buf)
        for i=0 to 3000:next
        msstat(x,y,bl,br)
    until bl=0
endfunc
func draw_sel_block()
    draw_selector()
    box(720,226,760,266,15)
    box(720,460,760,500,15)
```

```
    draw_close_white()
    draw_close_door()
endfunc
func draw_selector()
    int i,x,y,fp,col
    str s
    error off
    i=0
    for y=0 to 7
        for x=0 to 7
            s = "rmdat."+right$("00"+str$(i),3)
            fp = fopen(s,"r")
            if fp<>-1 then {
                exist_flag(i)=1:col=15
                freads(s,fp):locate x*10+4,y*3+3:print left$(s,8)
                fclose(fp)
            } else exist_flag(i)=0:col=9
            draw_selbox(x,y,col)
            i=i+1
        next
    next
    error on
endfunc
func draw_selbox(x;int,y;int,col;int)
    box(x*80+30,y*48+38,x*80+30+68,y*48+38+36,col)
endfunc
func draw_close_white()
    fill(721,227,759,265,0)
    box(730,247,750,262,15)
    box(735,237,745,247,15)
    box(732,249,747,257,15)
    line(737,237,737,247,15)
    line(739,237,739,247,15)
    line(741,237,741,247,15)
endfunc
func draw_open_white()
    fill(721,227,759,265,0)
    box(730,247,750,262,15)
    box(735,230,745,240,15)
    box(732,249,747,257,15)
```

```
    line(737,230,737,240,15)
    line(739,230,739,240,15)
    line(741,230,741,240,15)
    box(736,244,744,247,15)
    fill(738,240,742,244,15)
endfunc
```

●リスト……2　remocon. fnc

```
0000  48 55 00 00 00 00 00 00  : 9D
0008  00 00 00 00 00 00 01 3C  : 3D
0010  00 00 00 00 00 00 00 00  : 00
0018  00 00 00 26 00 00 00 1E  : 44
0020  00 00 00 00 00 00 00 00  : 00
0028  00 00 00 00 00 00 00 00  : 00
0030  00 00 00 00 00 00 00 00  : 00
0038  00 00 00 00 00 00 00 00  : 00
0040  00 00 00 40 00 00 00 40  : 80
0048  00 00 00 40 00 00 00 40  : 80
0050  00 00 00 40 00 00 00 40  : 80
0058  00 00 00 40 00 00 00 40  : 80
0060  00 00 00 42 00 00 00 52  : 94
0068  00 00 00 66 00 00 00 00  : 66
0070  00 00 00 00 00 00 00 00  : 00
0078  00 00 00 00 00 00 00 00  : 00
-----------------------------------
SUM:  48 55 00 CE 00 00 01 AC  1F60

0080  4E 75 72 6D 72 65 61 64  : 3E
0088  00 72 6D 77 72 69 74 65  : 0A
0090  00 00 00 00 00 5A 00 00  : 5A
0098  00 60 00 02 00 34 FF FF  : 94
00A0  00 02 00 34 FF FF 00 00  : 34
00A8  00 72 00 00 00 E4 00 00  : 56
00B0  00 00 20 6F 00 16 43 E8  : D0
00B8  00 0A 20 2F 00 0C 2F 09  : 9D
00C0  2F 00 61 06 50 8F 70 00  : E5
00C8  4E 75 42 A7 FF 20 58 8F  : B2
00D0  23 C0 00 00 00 6E 00 7C  : CD
```

```
00D8  07 00 20 2F 00 04 20 6F  : E9
00E0  00 08 53 80 6B 1A 22 7C  : FE
00E8  00 E9 A0 01 08 11 00 00  : A3
00F0  66 FA 10 D1 4E 71 4E 71  : BF
00F8  4E 71 4E 71 51 C8 FF F4  : 8A
----------------------------------
SUM:  A9 56 33 57 44 E6 9D 14  A4A1

0100  02 7C F8 FF 20 2F 00 04  : C8
0108  20 6F 00 08 02 10 00 01  : AA
0110  00 18 00 08 51 C8 FF F6  : 2E
0118  2F 39 00 00 00 6E FF 20  : F5
0120  58 8F 4E 75 20 6F 00 16  : 4F
0128  43 E8 00 0A 20 2F 00 0C  : 90
0130  2F 09 2F 00 61 06 50 8F  : AD
0138  70 00 4E 75 42 A7 FF 20  : 3B
0140  58 8F 23 C0 00 00 00 6E  : 38
0148  00 7C 07 00 20 2F 00 04  : D6
0150  20 6F 00 08 53 80 6B 14  : E9
0158  22 7C 00 E9 A0 07 12 98  : D8
0160  4E 71 4E 71 4E 71 4E 71  : FC
0168  51 C8 FF F4 02 7C F8 FF  : 81
0170  2F 39 00 00 00 6E FF 20  : F5
0178  58 8F 4E 75 00 00 00 04  : AE
----------------------------------
SUM:  4B B3 88 8E B9 D1 0F 9E  0ED2

0180  00 04 00 04 00 04 00 04  : 10
0188  00 04 00 04 00 04 00 04  : 10
0190  00 04 00 2A 00 04 00 10  : 42
0198  00 04 00 28 00 48 00 2A  : 9E
01A0  00 2E 02 01 00 00 00 8A  : BB
01A8  5F 72 6D 72 65 61 64 00  : DA
01B0  02 01 00 00 00 FC 5F 72  : D0
01B8  6D 77 72 69 74 65 00 00  : 98
01C0  00 00 00 00 00 00 00 00  : 00
01C8  00 00 00 00 00 00 00 00  : 00
01D0  00 00 00 00 00 00 00 00  : 00
01D8  00 00 00 00 00 00 00 00  : 00
01E0  00 00 00 00 00 00 00 00  : 00
```

```
01E8  00 00 00 00 00 00 00 00  : 00
01F0  00 00 00 00 00 00 00 00  : 00
01F8  00 00 00 00 00 00 00 00  : 00
-----------------------------------
SUM:  CE 28 E1 36 D9 16 C3 3E  4999
```

●リスト……3　赤外線リモコンサポート関数

```
*****************************************
*                                       *
*      赤外線リモコンサポート関数        *
*                                       *
*  1989-02-24  Programmed by M.kuwano   *
*                  No rights reserved   *
*****************************************
*      rmread(size,buffer)              *
*          unsigned short size;         *
*          unsigned char  *buffer;      *
*.......................................*
*      rmwrite(size,buffer)             *
*          unsigned short size;         *
*          unsigned char  *buffer;      *
*.......................................*
*                                       *
* 外部関数／ライブラリ共通です          *
*                                       *
* ＢＡＳＩＣの外部関数にするときは      *
*      as  remocon.s                    *
*      lk  /o remocon.fnc remocon.o     *
*                                       *
* コンパイルするときは                  *
*      cc  wave.bas remocon.o           *
*                                       *
* のようにしてください                  *
*                                       *
*****************************************

                .include        doscall.mac
```

```
		.include	fdef.h
		.globl		_rmread
		.globl		_rmwrite
		.text
		.even
*
* インフォーメーション・テーブル
*
		dc.l		X_INIT
		dc.l		X_RUN
		dc.l		X_END
		dc.l		X_SYS
		dc.l		X_BRK
		dc.l		X_CTRL_D
		dc.l		X_RES1
		dc.l		X_RES2
		dc.l		PTR_TOKEN
		dc.l		PTR_PARAM
		dc.l		PTR_EXEC
		dc.l		0,0,0,0,0

X_INIT:
X_RUN:
X_END:
X_SYS:
X_BRK:
X_CTRL_D:
X_RES1:
X_RES2:
		rts

*
* ファンクション名テーブル
*
PTR_TOKEN:
		dc.b		'rmread',0
		dc.b		'rmwrite',0
		dc.b		0

		.even
```

```
*
* パラメータ・テーブル
*
PTR_PARAM:
                dc.l            RMREAD_PAR
                dc.l            RMWRITE_PAR

*
* パラメータＩＤテーブル
*
RMREAD_PAR:
                dc.w            int_val
                dc.w            aryl_c
                dc.w            void_ret
RMWRITE_PAR:
                dc.w            int_val
                dc.w            aryl_c
                dc.w            void_ret

*
* 関数アドレステーブル
*
PTR_EXEC:
                dc.l            rmbread
                dc.l            rmbwrite


*
* スタック・バッファ
*
SPBUF:
                ds.l            1


                .even
*
* リモコンデータ読みだし
*
JOYPORT         equ     $e9a001
rmbread:
```

```
		movea.l		22(sp),a0
		lea.l		10(a0),a1
		move.l		12(sp),d0
		move.l		a1,-(sp)
		move.l		d0,-(sp)
		bsr		_rmread
		addq.l		#8,sp
		moveq.l		#0,d0
		rts

_rmread:
		clr.l		-(sp)
		dc.w		_SUPER
		addq.l		#4,sp
		move.l		d0,SPBUF

		ori.w		#$0700,sr		*	disable_trap
();

		move.l		4(sp),d0
		movea.l		8(sp),a0

		subq.l		#1,d0
		bmi		rmread_end

		movea.l		#JOYPORT,a1
rmread_wait:
		btst		#0,(a1)
		bne		rmread_wait
rmread_loop:
		move.b		(a1),(a0)+
		nop
		nop
		nop
		nop
		dbra		d0,rmread_loop

rmread_end:
		andi		#$f8ff,sr		*	enable_trap
();
```

```
		move.l		4(sp),d0
		movea.l		8(sp),a0
	rmread_conv:
		andi.b		#$01,(a0)
		ori.b		#$8,(a0)+
		dbra		d0,rmread_conv

		move.l		SPBUF,-(sp)
		dc.w		_SUPER
		addq.l		#4,sp

		rts

*
* リモコンデータ出力
*
STBPORT		equ	$e9a007
rmbwrite:
		movea.l		22(sp),a0
		lea.l		10(a0),a1
		move.l		12(sp),d0
		move.l		a1,-(sp)
		move.l		d0,-(sp)
		bsr		_rmwrite
		addq.l		#8,sp
		moveq.l		#0,d0
		rts

	_rmwrite:
		clr.l		-(sp)
		dc.w		_SUPER
		addq.l		#4,sp
		move.l		d0,SPBUF

		ori.w		#$0700,sr		*	disable_trap
();

		move.l		4(sp),d0
		movea.l		8(sp),a0
```

```
            subq.l          #1,d0
            bmi             rmwrite_end

            movea.l         #STBPORT,a1
    rmwrite_loop:
            move.b          (a0)+,(a1)
            nop
            nop
            nop
            nop
            dbra            d0,rmwrite_loop

    rmwrite_end:
            andi            #$f8ff,sr               *       enable_trap
();

            move.l          SPBUF,-(sp)
            dc.w            _SUPER
            addq.l          #4,sp

            rts

            .end
```

CRT切り替え機

パソコンが複数になってきたとき，あると便利なのがCRT切り替え機です。市販品もありますが，ここでは本体の電源ONやキー操作で自動的に切り替わるなど，X68000の周辺機器らしさを出すようにしてみました。

すでにX68000を持っていて，もう1台新機種を買い足そうと考えているようなときに便利なのがCRT切り替え機です。1台のCRTを2台のX68000で共用するようにすれば，ディスプレイ分のお金を節約できますし，机の上も有効に利用できるようになります。このような目的のため，アナログRGBディスプレイの信号を切り替えるようにしたものも市販されています。さすがにメーカー製だけに外観は見栄えよくできていますが，中身はアナログRGBコネクタの信号をスイッチで切り替えるだけのものであり，切り替え操作は手動でスイッチ操作をするしかありません。

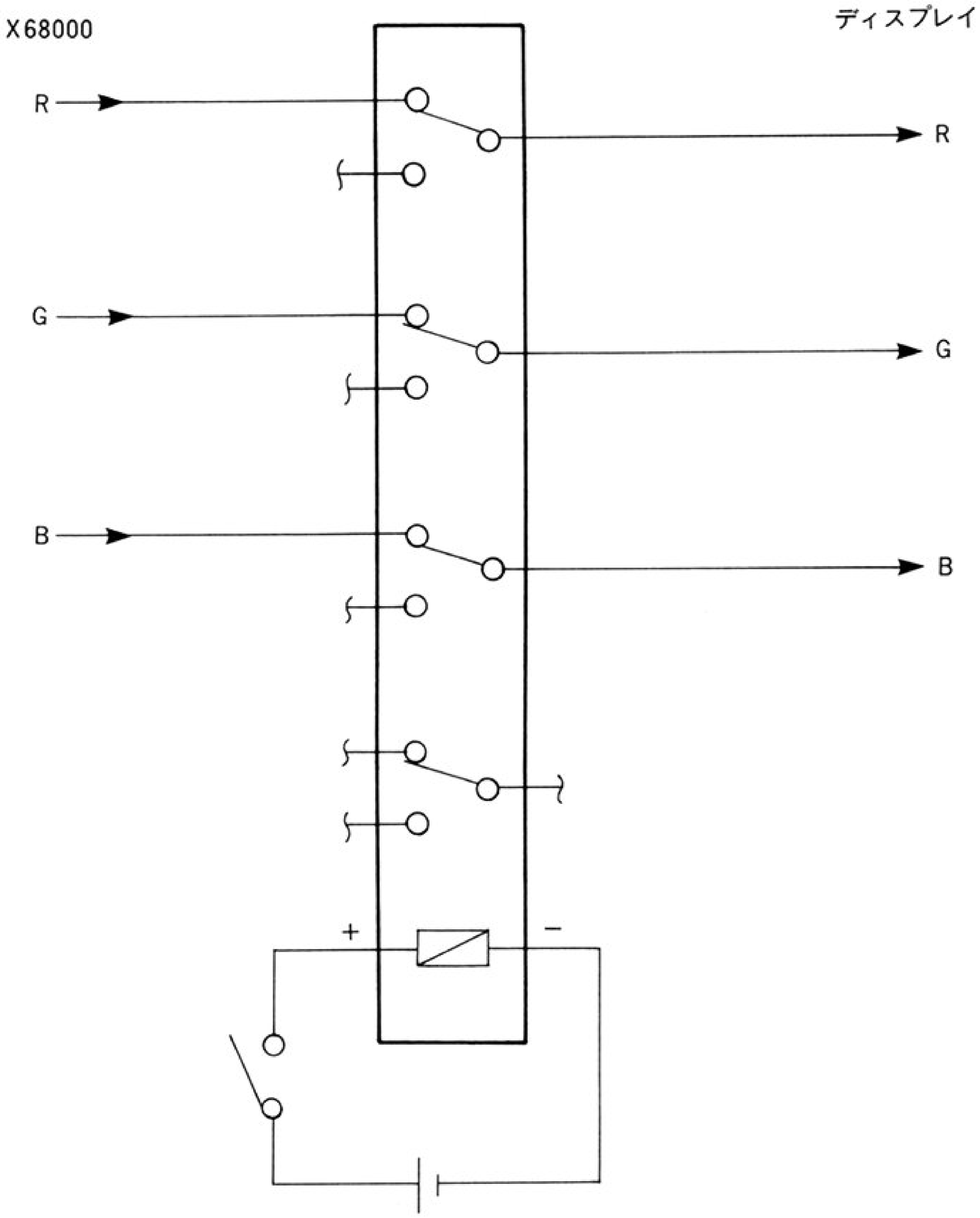
せっかくディスプレイ制御が本体から行え，チャンネル切り替えまでキーボードからできるようになっているX68000なのですから，ディスプレイ切り替えもせめて本体の電源を入れれば自動的にそちらに切り替わるくらいの芸がほしいところでしょう。

ここでは，スイッチのかわりにリレーを使うことで，単なるマニュアル切り替えだけでなく，本体の電源を入れれば自動的にディスプレイが切り替わり，さらにキーボード操作でもディスプレイを使いたい側に向けられるようにした切り替え機を作ってみることにします。

●1 切り替えの実験

まず，アナログRGB信号をリレーで切り替えられるかの実験をしてみました。アナログRGB信号は音声やディスプレイ制御信号などよりも格段に広い周波数帯域を使用するうえ，

●図……1　リレーによる切り替え確認



信号波形に乱れがあると画面がぼやけたり色ずれとなって目に見えてしまいます。一般的なリレーではたしてうまく切り替えられるか実験してみることにしました。使用したのはオムロンのＧ６Ａシリーズの4回路入りのものです。比較的安価で，しかもリレーを扱っている店ならたいてい置いてあるようですので入手は容易でしょう。接点が完全に密封されていて，中に不活性ガスが封入されているので，寿命の点でも安心できます。

図1のように，リレーの3回路分を使って X68000 本体から出力されるアナログ RGB の信号を ON/OFF してみます。直結したときと，リレーを通したときで映像に目立った変化はないか，画像データを表示させて色むらや色ずれ，色の変化などが起こらないか，リレーを OFF したときでもぼんやり絵が見えたりしないかなど，気になっていた点を中心にみてみましたが，目でわかるような変化は見受けられません。X68000 のアナログ RGB 信号程度ならこのリレーで十分に使い物になるようです。

●2 切り替え回路の検討

切り替えの心臓部であるリレーについての問題はクリアできたので，次に実際の切り替え回路を考えてみることにします。本体の電源 ON/OFF に自動的に連動させようとすると，ジョイスティックポートなどに出ている＋5 V や，ディスプレイの同期信号を検出する，ディスプレイ制御信号を利用するなどの方法が考えられます。このうち，ディスプレイ制御信号は，キーボードによって発生させることもできますから，うまく利用すれば電源 ON/OFF だけでなく，キーボード操作によって操作したい側にディスプレイを切り替えることも簡単に実現できそうです。しかも，ディスプレイ制御信号コネクタには＋5 V も出力されていますので，リレー駆動用の電源もとることができます。

次にリレー切り替えの回路を考えます。ディスプレイ制御信号を解釈して特定のコードにだけ反応するようにするのはかなり難しいことですが，単純にディスプレイ制御信号がきたらそちらに切り替わるようにするだけなら簡単です。一応，マニュアルでも切り替えられるようにスイッチもつけておきましょう。ディスプレイ制御信号は両方からきたものをそのまま合成してしまえば，どちら側からもディスプレイ制御が可能となります。

このようにした場合，2台の X68000 とも電源を入れている状態でどちらか片方の電源を OFF にすると切り替え機は OFF にした側に向いてしまい，ディスプレイの電源も OFF になってしまうのが唯一の欠点です。この場合だけはワイヤレスリモコンとマニュアルスイッチを

使うか，ちょっとしたプログラムを作って本体から電源 ON コードを発生させることで対処することにしましょう。

3 回路設計

図 2 が，製作した切り替え機の全回路図です。簡単に各回路ブロックの説明をしておきましょう。

3·1 ディスプレイ選択部

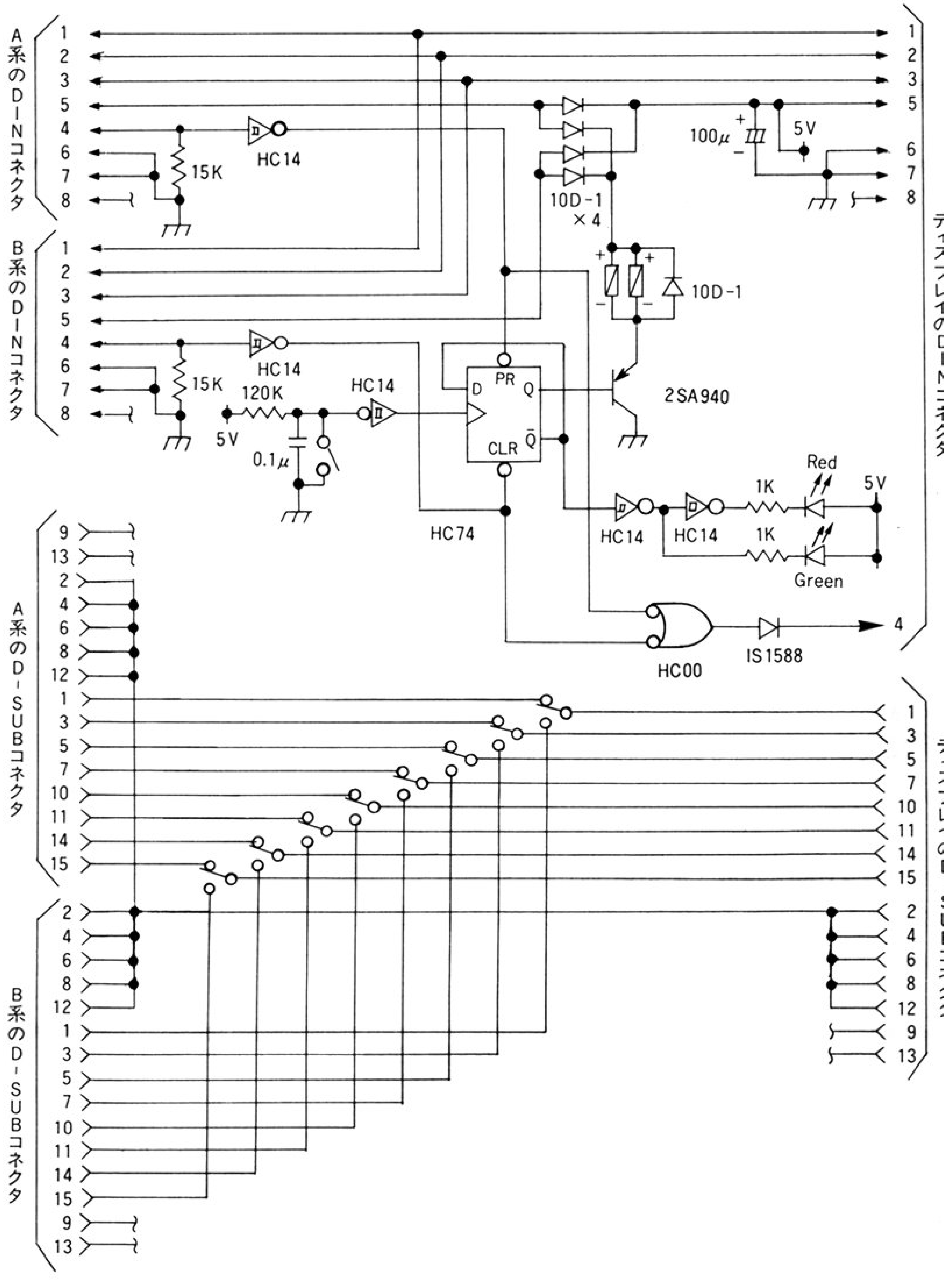
ディスプレイ制御信号がきた側に切り替わり，さらにスイッチで交互に切り替わる（トグル動作と呼ぶことにします）ようにするため，HC74 を 1 つ使いました。フリップフロップのセットとリセットにそれぞれA系，B系のディスプレイ制御信号を入れてディスプレイ制御信号がきた側に向くようにしておき，さらにクロック入力にスイッチを取り付けて，押すたびに FF の出力が反転するようにしておきます。

X68000 のディスプレイ制御信号の出力は TTL(LS11)の出力にダイオードを入れて，H レベルのときだけ電流が流れるようになっていますので，プルダウン抵抗とシュミットトリガタイプの IC (HC14) で受けておきます。LS11 の H レベル出力は最小でも 2.7 V なのに対して HC14 の H レベルスレッショルドは平均で 2.4 V，最大で 3 V 程度です。運悪く両方とも最悪値になるとまずいのですが，LS と HC を直結して問題が起きた経験はないのでこのままにしてあります。どうしても気になる方は HCT シリーズのゲートで受けるかオペアンプなどでコンパレータを作れば確実でしょう。

3·2 ディスプレイ制御信号合成部

切り替え機から出力されているディスプレイ制御信号は，単純にそれぞれの系からの出力信号の OR をとっています。使うのは一人ですから，両方から同時に制御信号が出ることはほとんどありませんので，これでも十分でしょう。切り替え機からの出力の回路は，本体の回路を

●図……2　CRT 切り替え機回路図



まねてダイオードを入れています。デバイスにHC00を使っておりHレベル出力電圧の最小値がLSシリーズよりも1V以上高いので，ダイオードにはショットキーバリアタイプではなく，一般的な小型のシリコンダイオードを使ってみました。念のためシンクロスコープで出力波形を確認しておきましたが，特に問題はないようです。

❸·3 リレードライブ部

最後にリレーのコイルをドライブする部分を考えます。コイルの抵抗を実測してみると約66Ωでした。電源を5Vとすると約76mA，2つで152mA流れることになります。74シリーズの論理ICのドライブ能力はせいぜい数十mAです。直接リレーをドライブするのはとうてい無理なのでトランジスタを1つ追加します。トランジスタは2SAや2SBタイプで最大コレクタ電流が200mA以上，hfe(直流電流増幅率)が50以上のものならなんでもかまいません。

●4 製作上の注意点

さほど複雑な回路ではありませんので，テスターなどで配線チェックをしながら製作しておけばまず動かないということはないでしょうが，一応，製作上注意しておきたい点などについてまとめておきます。

❹·1 アナログRGB信号の配線

アナログRGB信号は周波数の高いアナログ信号であるうえ，波形の乱れが画質に直接影響しますので配線は同軸ケーブルを用い，長さもできるだけ同じになるようにしてください。アナログRGB信号のインピーダンスは75Ωです。使用する同軸ケーブルも1.5C-2Vなど同じ75Ωのものがよいのですが，なければ50Ω系の1.5D-2Vなどでもかまいません。私の作ったものでは他の信号線まですべて同軸を使用していますが，ここまでやる必要はありません。適当に余った線材やフラットケーブルを引き裂いたものを流用しておけばよいでしょう。

❹・2 リレーの極性に注意

リレーのコイルの極性は間違えやすいので気をつけてください（このタイプのリレーは，感度を上げるためか有極になっています）。リレー上部に書いてある図はリレーを裏から見たときの図なので，上から見た図だと思って配線すると極性がさかさまになってしまいます。不安な方は電池などを使って極性を確認してから配線にかかるとよいでしょう。

❹・3 LEDについて

切り替え表示のLEDは何でもかまいませんが，緑と赤というように色を分けておいたほうが見やすくなります。昔はLEDといえば赤と緑くらいしかありませんでしたが，今では黄やオレンジ，青などもありますので，好みと懐具合に応じて好きなものを使えばよいでしょう。

❹・4 ケースの選択

ケースはX68000の横に置くということから縦型のものを探したのですが，手頃な大きさのものがなかなか見つかりません。まあ，なんとか縦に置いてもおかしくなさそうなものということで見つけたのが，タカチ電機工業のSY-190という型番のものです。ケースはプラスチック（ABS樹脂のようです）で，前面と背面のパネルはアルミ板です。ケースを止めるネジの頭が表に出っ張りませんのでX68000の横に置いても傷をつける心配がありません。色はアイボリー（白）と黒の2種類があります。ケースの大きさは190×54×200 mmと，基板サイズに比べてずいぶん大きく感じますが，背面パネルにD-SUBコネクタと8ピンのDINコネクタをそれぞれ3つずつ付けるのでこのくらいのパネル面積は絶対に必要です。また，内部配線に同軸を使っているためどうしても小回りがききませんから，このくらいの大きさがあったほうが組み込みが楽にできます。

なお，ケース内は同軸で混雑するので，組み立てや動作チェックまでは外部ですませておき，最後にケースに組み込むようにしたほうが楽に作業できます。

❹・5 ディスプレイ接続ケーブル

X68000本体と切り替え機の間のケーブルは本体に付属してきたものをそのまま使えばよいのですが，切り替え機とディスプレイの間のケーブルは新規に入手しなくてはなりません。ディスプレイ制御信号用にDINの8ピンコネクタ同士の1対1ケーブル，アナログRGB信号用に15ピンD-SUBコネクタの1対1ケーブルが必要です。どちらもパソコンショップなどで入手することができると思います。私はDINケーブルのほうはジャンク屋で300円で売っていたものを，D-SUBケーブルのほうは試しにフラットケーブルを使って自作してみました。信号同士の干渉を抑えるため，かならず1本おきにGNDがくるようにしてください。アナログRGB信号をフラットケーブルでつなぐことにかなり不安があったのですが，距離が短かったせいかほとんど問題なく使用できました。ただし，この状態では周りにノイズをばらまき放題になっているのは確実ですし，外来ノイズにも弱いはずです。できるだけきちんとしたケーブルを購入して使うべきでしょう。

●5 試用

でき上がった切り替え機を実際に使用してみました。A系のX68000の電源を入れるとLEDが点灯し，ディスプレイにはいつものタイトルがドーンと真ん中に現れます。そこでB系の電源をON。切り替え機のリレーが小さく　カチ！　と鳴って画面が切り替わります。そこでA系のキーボードでSHIFT＋テンキーの1などとすると，再びリレーが動いてA系の画面。当然，ディスプレイからの音声出力もちゃんと画面に連動して切り替わります。使いたい側の電源を入れるだけで自動的にディスプレイが切り替わるので，最近では切り替え機の存在をつい忘れてしまいそうになっています。

6 おわりに

1台のディスプレイを2台のX68000で共用することができるようになりました。自分で必要にせまられて作ったものではありますが，実際に使ってみるとこんなに便利なものだとは思いませんでした。もう少し工夫すると3台以上のX68000でディスプレイを共有させることもできるのですが，それは宿題ということにしておきましょう。

おわりに

早いもので，『Inside X68000』が世にでてから一年がたちました。私が望んでいたような形でのハードウェア解説本は『Inside』と『Outside』の2冊をもってひとまず完結ということになります。

この間，数多くのお便りをいただきました。そのなかで「6,800円」という価格はシャレのつもりでつけたのではないかとか，「Inside...」という名称がアップル社のマッキントッシュの有名な解説本と同じであることを指摘された方も少なからずおられましたので，ここで真相を打ち明けておきましょう。

価格については，当初確かに冗談でいっていたことがあったのですが，実際に本になってみると，ページ数もさることながら，多量の図版のほとんどが手書き原稿で，それらすべてをトレス屋さんに出したことなどから制作にずいぶんお金がかかってしまい，6千円から7千円程度にしないと採算がとれそうにない状態になってしまったということです。

また，書名については，X68000のハードウェアの本を書かないかという話が私のところにきた直後にさかのぼります。とりあえずこれまでほしいと思っていたハードウェア情報を書き出していったのですが，あまりにも内容が多くなりすぎて収拾がつかなくなってしまいました。そこでひとまずOSやデバイスドライバなど，本体のLSIに直接アクセスするようなソフトウェアを書く人のための情報と，オプションボードや周辺機器などを自作しようとする人のための情報に大きく二分し，前者はやや内部よりなので「Inside X68000」，後者は本体の外まで伸びていく部分なので「Outside X68000」という仮名をつけていたのがそのまま書名となったのです。つまり，InsideとOutsideは2つで1つのX68000のハードウェア解説本なのです。

ソフトとハードは車の両輪であるといわれてずいぶん時間がたちました。CPUの速度などはさしずめ回転数といったところでしょうか。確かに回転数が上がれば前進する速度はあがりますが，残念ながら最近はハードウェアが硬直化し，ソフトの方ばかりが大きくなってきているように思えます。片方の車輪ばかり大きくなっては回転数がいくら上がったところで同じところをぐるぐる回るばかりで決して前に進むことはできません。ソフトばかりが大きくなっても回転半径が小さくなる一方です。

このことは個人レベルについてもあてはまるのではないでしょうか。他機種ではみられない

ような数々の機能を標準装備したうえ，ユーザにそれらの直接の操作を解放した X68000 というハードウェアが登場した当初，ソフトウェア環境はきわめて貧弱なものでした。しかし，そのハードウェアは，私たちがこれまでのパソコンでは決して見ることのできなかった夢を抱かせてくれ，いまでは CPU 単体の速度 (ベンチマークでは同一クロックの 8086 や V 30 といい勝負です。20 MHz の 486 と比較すると 16 分の 1 程度ということになるでしょうか)からは想像もつかないようなソフトウェア環境を得るに至っています。ゲーム類のみならず，486+MS-Windows よりもはるかに快適な GUI 環境を三世代以上も前の CPU と描画機能すら持たない CRT コントローラで実現してしまった「SX-WINDOW」などは CPU ブランド信奉者にとっては許しがたい存在でしょう。

もちろん，ここまできてもまだまだハードウェアが使い尽くされたとはいえないでしょうし，これからもソフトウェアからの機能強化は図られていくでしょう。しかし，使用する方面によってはそろそろ X68000 の標準機能だけでは物足りなくなってきていることも事実です。拡張のためのオプションボードなども売られていますが，それらをもってしてもユーザの多種多様な発想にすべてこたえるのはとうてい不可能です。

そんなときに，ハードウェアの限界だから仕方がないとあきらめたり，機種変更に走るのも一つの選択ですが，そこでソフトウェアだけでは実現できない部分を助けるエッセンスのようなハードウェアを追加するというのも，個人が自由に扱えるパーソナルコンピュータならではの選択枝ではないでしょうか。

パソコンをその与えられたハードウェアだけの閉じたものと見ず，外部世界と広くわたりあえるインテリジェントマシンと考えれば，もっと広い楽しみ方が生まれるのではないでしょうか。

本書が単なる回路図集にとどまらず，同じような思いを抱かれている方々の参考資料としてお役に立つことを願ってやみません。

1993 年 2 月 27 日　　伝説巨人イデオンを鑑賞しつつ

くわのまさひこ
桑野雅彦

参考文献

マルチチップ ……日本電気
mcs YM3802 アプリケーションマニュアル ……日本楽器製造
本体，各ボード類のアプリケーションマニュアル ……シャープ
MC68000 マイクロプロセッサユーザーズマニュアル ……CQ 出版社
16 ビットマイクロプロセッサ TLCS-68000 マイクロプロセッサ編 ……東芝
Inside X68000 ……ソフトバンク

INDEX

H

I

J

K

L

M

N

P

R

S

U

V

W

●
●
●
Outside X68000
●

● 1993年4月24日　初版第1刷印刷
● 1993年4月30日　初版第1刷発行
●
● 著　者　桑野雅彦（くわのまさひこ）
● 発行者　橋本五郎
● 発行所　ソフトバンク株式会社　出版事業部
● 〒103　東京都中央区日本橋浜町3-42-3
● 営業部　☎03(5642)8101
● 編集部　☎03(5642)8140
● 印刷所　壮光舎印刷株式会社

● ISBN4-89052-395-2　C0055

ISBN4-89052-395-2 C0055 P3900E

SOFT BANK ソフトバンク
定価…3,900円［本体3,786円］

Outside X68000

本書は、シャープのX68000で利用できるオプションボードの回路図や
インタフェース部の回路、ブロック図、ピン設定などの
ハードウェア情報をまとめたテクニカルデータブックです。
また、拡張ボードの機械的な仕様（寸法）やX68000本体の拡張スロットに関する
DC規格、各信号の意味やそれぞれの動作タイミングなど、
拡張スロットを利用するうえで必要となる情報も盛り込みました。
さらに、著者が製作した周辺機器を例に、
実際に周辺機器を自作するためのノウハウも紹介しています。